

On The Reachability Problem for Recursive Hybrid Automata with One and Two Players

Shankara Narayanan Krishna, Lakshmi Manasa, and Ashutosh Trivedi

Indian Institute of Technology Bombay, India,
krishnas,manasa,trivedi@cse.iitb.ac.in

Abstract. Motivated by the success of bounded model checking framework for finite state machines, Ouaknine and Worrell proposed a time-bounded theory of real-time verification by claiming that restriction to bounded-time recovers decidability for several key decision problem related to real-time verification. In support of this theory, the list of undecidable problems recently shown decidable under time-bounded restriction is rather impressive: language inclusion for timed automata, emptiness problem for alternating timed automata, and emptiness problem for rectangular hybrid automata. The objective of our study was to recover decidability for general recursive timed automata—and perhaps for recursive hybrid automata—under time-bounded restriction in order to provide an appealing verification framework for powerful modeling environments such as Stateflow/Simulink. Unfortunately, however, we answer this question in negative by showing that time-bounded reachability problem stays undecidable for recursive timed automata with five or more clocks. While the bad news continues even when one considers well-behaved subclasses of recursive hybrid automata, we recover decidability by considering recursive hybrid automata with bounded context using a pass-by-reference mechanism, or by restricting the number of variables to two, with rates in $\{0, 1\}$.

1 Introduction

Recursive state machines (RSMs), as introduced by Alur, Etessami, and Yannakakis [6], are a variation on various visual notations to represent hierarchical state machines, notably Harel’s statecharts [15] and Object Management Group supported UML diagrams [19], that permits recursion while disallowing concurrency. RSMs closely correspond [6] to pushdown systems [9], context-free grammars, and Boolean programs [7], and provide a natural specification and verification framework to reason with sequential programs with recursive procedure calls. The two fundamental verification questions for RSM, namely reachability and Büchi emptiness checking, are known to be decidable in polynomial time [6,13].

Hybrid automata [4,3] extend finite state machines with continuous variables that permit a natural modeling of hybrid systems. In a hybrid automaton the variables continuously flow according to a given set of ordinary differential equations within each discrete states, while they are allowed to have discontinuous jumps during transitions between states that are guarded by constraints over variables. In this paper we study the reachability problem for recursive hybrid automata that generalize recursive state machines with continuous variables, or equivalently hybrid automata with recursion.

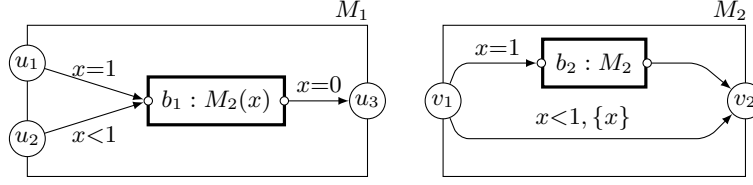


Fig. 1. An example of recursive timed automata with one clock and two components

In this paper we restrict our attention to so-called *singular hybrid automata* where the dynamics of every variable is restricted to state-dependent constant rates. A variable is often called a *clock* if its rate over all the states is 1, while it is called a *stopwatch* if its rate is either 0 (clock is stopped) or 1 (clock is ticking) in different states. A timed automaton [4] is a hybrid automaton where all the variables are clocks, while a stopwatch automaton [16] is a hybrid automaton where all the variables are stopwatches. It is well known that the reachability problem is decidable (PSPACE-complete) for timed automata [4] and undecidable for stopwatch automata with 3 stopwatches [11].

Trivedi and Wojtczak [20] introduced *recursive timed automata* (RTAs) as an extension of timed automata with recursion to model real-time software systems. Formally, an RTA is a finite collection of components where each component is a timed automaton that in addition to making transitions between various states, can have transitions to “boxes” that are mapped to other components modeling a potentially recursive call to a subroutine. During such invocation a limited information can be passed through clock values from the “caller” component to the “called” component via two different mechanism: a) *pass-by-value*, where upon returning from the called component a clock assumes the value prior to the invocation, and b) *pass-by-reference*, where upon return clocks reflect any changes to the value inside the invoked procedure.

Example 1. The visual presentation of a recursive timed automaton with two components M_1 and M_2 , and one clock variable x is shown in Figure 1 (example taken from [20]), where component M_1 calls component M_2 via box b_1 and component M_2 recursively calls itself via box b_2 . Components are shown as thinly framed rectangles with their names written next to upper right corner. Various control states, or “nodes”, of the components are shown as circles with their labels written inside them, e.g. see node u_1 . Entry nodes of a component appear on the left of the component (see u_1), while exit nodes appear on the right (see u_3). Boxes are shown as thickly framed rectangles inside components labeled $b : M(C)$, where b is the label of the box, M is the component it is mapped to, and C is the set of clocks passed to M by value and the rest of the variables are passed by reference. When the set C is empty, we just write $b : M$ for $b : M(\emptyset)$. Each transition is labelled with a guard and the set of reset variables, (e.g. transition from node v_1 to v_2 can be taken only when variable $x < 1$, and after taking this transition, variable x is reset). To minimize clutter we omit empty reset sets.

Trivedi and Wojtczak [20] showed that the reachability and termination (reachability with empty calling context) problem is undecidable for RTAs with three or more clocks. Moreover, they considered the so-called glitch-free restriction of RTAs—where at each invocation either all clocks are passed by value or all clocks are passed by reference—and showed that the reachability (and termination) is EXPTIME-complete for RTAs with two or more clocks. In the model of [20] it is compulsory to pass all

	Recursive Timed Automata		Recursive Hybrid Automata	
	TUB	TB	TUB	TB
Pass by reference	D	D	$\mathbf{U} (\geq 3 \text{ sw})$ [11]	\mathbf{D} (Bounded context)
Pass by value	D	D	$\mathbf{U} (\geq 3 \text{ sw})$	$\mathbf{U} (\geq 14 \text{ sw})$
Glitch free	D	D	$\mathbf{U} (\geq 3 \text{ sw})$ $\mathbf{D} (\leq 2 \text{ sw})$	$\mathbf{U} (\geq 14 \text{ sw})$ $\mathbf{D} (\leq 2 \text{ sw})$
Unrestricted	$\mathbf{U} (\geq 3 \text{ clocks})$	$\mathbf{U} (\geq 5 \text{ clocks})$	$\mathbf{U} (\geq 2 \text{ sw})$	$\mathbf{U} (\geq 5 \text{ sw})$

Table 1. Summary of results related to RHA - single player game. Results shown in bold are contributions from this paper, while results shown in gray color are contributions from [20]. Here TUB and TB stand for time-unbounded and time-bounded reachability problems, respectively. U stands for undecidable, D for decidable, and sw for stopwatches.

	Recursive Timed Automata		Recursive Stopwatch Automata	
	TUB	TB	TUB	TB
Glitch free	D	D	$\mathbf{U} (\geq 3 \text{ sw})$ $\mathbf{D} (\leq 2 \text{ sw})$	$\mathbf{U} (\geq 4 \text{ sw})$ $\mathbf{D} (\leq 2 \text{ sw})$
Unrestricted	$\mathbf{U} (\geq 2 \text{ clocks})$	$\mathbf{U} (\geq 3 \text{ clocks})$	$\mathbf{U} (\geq 2 \text{ sw})$	$\mathbf{U} (\geq 3 \text{ sw})$

Table 2. Summary of results for two-player games. Results shown in bold are contributions from this paper, while results shown in gray color are from [20].

the clocks at every invocation with either mechanism. Abdulla, Atig, and Stenman [1] studied a related model called timed pushdown automata where they disallowed passing clocks by value. On the other hand, they allowed clocks to be passed either by reference or not passed at all (in that case they are stored in the call context and continue to tick with the uniform rate). It is shown in [1] that the reachability problem for this class remains decidable (EXPTIME-complete). In this paper we restrict ourselves to the recursive timed automata model as introduced in [20].

Contributions. In this paper we consider time-bounded reachability problem for RTA and show that the problem stays undecidable for RTA with 5 or more clocks. We also consider the extension of RTAs to recursive hybrid automata (RHAs) and show that the reachability problem stays undecidable even for glitch-free RHAs with 3 or more stopwatches (clocks that can be paused), while we show decidability of glitch-free RHAs with 2 stopwatches. We also show that the reachability problem is undecidable for unrestricted RHA with two or more stopwatches. For the time-bounded reachability case, we show that the problem stays undecidable even for glitch-free variant of RHAs with 14 or more stopwatches. On the positive side, we show decidability of time-bounded reachability in glitch-free RHAs where with pass-by-reference only mechanism. Our results are summarized and compared with known results in Table 1.

We study these problems for two player games on RTA and RHA also. The undecidability saga continues even for games with lesser number of clocks than single-player case. The results for games have been summarized in Table 2.

Related work. For a survey of models related to recursive timed automata and dense-time pushdown automata we refer the reader to [20] and [1]. Another closely related model is introduced by Benerecetti, Minopoli, and Peron [8] where pushdown automata is extended with an additional stack used to store clock valuations. The reachability problem is known to be undecidable for this model. We do not consider this model in the current paper, but we conjecture that time-bounded reachability problem for this model is also undecidable.

Two special kinds of RSMs with restricted recursion are hierarchical RSMs and bounded stack RSMs [12]. [12] gives efficient algorithms for the reachability analysis of hierarchical and bounded stack RSMs, and algorithms for the latter might be useful in the analysis of programs without infinite recursion. The language theory of bounded context recursion has been studied recently [17]. Hierarchical hybrid systems studied by Alur et al. [5] are a restriction of recursive hybrid automata where recursion is disallowed but concurrency is allowed. A number of case-studies with the tool CHARON [5] demonstrate the benefit of hierarchical modeling of hybrid systems.

Organization. In the next section, we begin by reviewing the definition of recursive state machines followed by a formal definition of recursive (singular) hybrid automata. We also formally define the termination and the reachability problems for two players on this model, and present our main results. Finally, Sections 4 and 5 details our main undecidability results while Sections 6 and 7 discuss our decidability results.

2 Preliminaries

2.1 Reachability Games on Labelled Transition Systems.

A *labeled transition system* (LTS) is a tuple $\mathcal{L} = (S, A, X)$ where S is the set of *states*, A is the set of *actions*, and $X : S \times A \rightarrow S$ is the *transition function*. We say that an LTS \mathcal{L} is *finite (discrete)* if both S and A are finite (countable). We write $A(s)$ for the set of actions available at $s \in S$, i.e., $A(s) = \{a : X(s, a) \neq \emptyset\}$. A *game arena* G is a tuple $(\mathcal{L}, S_{\text{Ach}}, S_{\text{Tor}})$, where $\mathcal{L} = (S, A, X)$ is an LTS, $S_{\text{Ach}} \subseteq S$ is the set of states controlled by player Achilles, and $S_{\text{Tor}} \subseteq S$ is the set of states controlled by Tortoise. Moreover, sets S_{Ach} and S_{Tor} form a partition of the set S . In a *reachability game* on G rational players—Achilles and Tortoise—take turns to move a token along the states of \mathcal{L} . The decision to choose the successor state is made by the player controlling the current state. The objective of Achilles is to eventually reach certain states, while the objective of Tortoise is to avoid them forever.

We say that $(s, a, s') \in S \times A \times S$ is a transition of \mathcal{L} if $s' = X(s, a)$ and a *run* of \mathcal{L} is a sequence $\langle s_0, a_1, s_1, \dots \rangle \in S \times (A \times S)^*$ such that (s_i, a_{i+1}, s_{i+1}) is a transition of \mathcal{L} for all $i \geq 0$. We write $\text{Runs}^{\mathcal{L}}$ ($\text{FRuns}^{\mathcal{L}}$) for the sets of infinite (finite) runs and $\text{Runs}^{\mathcal{L}}(s)$ ($\text{FRuns}^{\mathcal{L}}(s)$) for the sets of infinite (finite) runs starting from state s . For a set $F \subseteq S$ and a run $r = \langle s_0, a_1, \dots \rangle$ we define $\text{Stop}(F)(r) = \inf \{i \in \mathbb{N} : s_i \in F\}$. Given a state $s \in S$ and a set of final states $F \subseteq S$ we say that a final state is reachable from s_0 if there is a run $r \in \text{Runs}^{\mathcal{L}}(s_0)$ such that $\text{Stop}(F)(r) < \infty$. A strategy of Achilles is a partial function $\alpha : \text{FRuns}^{\mathcal{L}} \rightarrow A$ such that for a run $r \in \text{FRuns}^{\mathcal{L}}$ we have that $\alpha(r)$ is defined if $\text{last}(r) \in S_{\text{Ach}}$, and $\alpha(r) \in A(\text{last}(r))$ for every such r . A

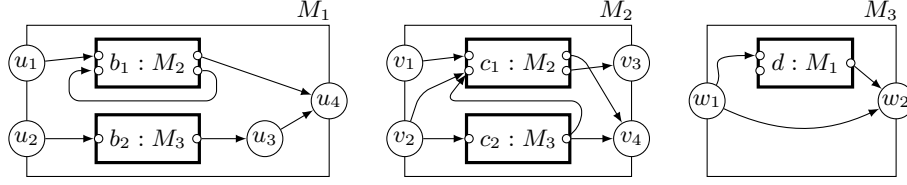


Fig. 2. Example recursive state machine taken from [2]

strategy of Tortoise is defined analogously. Let $\Sigma_{\text{Ach}}^{\mathcal{L}}$ and $\Sigma_{\text{Tor}}^{\mathcal{L}}$ be the set of strategies of Achilles and Tortoise, respectively. The unique run $\text{Run}(s, \alpha, \tau)$ from a state s when players use strategies $\alpha \in \Sigma_{\text{Ach}}^{\mathcal{L}}$ and $\tau \in \Sigma_{\text{Tor}}^{\mathcal{L}}$ is defined in a straightforward manner.

For an initial state s and a set of final states F , the lower value $\text{Val}_F^{\mathcal{L}}(s)$ of the reachability game is defined as the upper bound on the number of transitions that Tortoise can ensure before the game visits a state in F irrespective of the strategy of Achilles, and is equal to $\sup_{\tau \in \Sigma_{\text{Tor}}^{\mathcal{L}}} \inf_{\alpha \in \Sigma_{\text{Ach}}^{\mathcal{L}}} \text{Stop}(F)(\text{Run}(s, \alpha, \tau))$. The concept of upper value is $\overline{\text{Val}}_F^{\mathcal{L}}(s)$ is analogous and defined as $\inf_{\alpha \in \Sigma_{\text{Ach}}^{\mathcal{L}}} \sup_{\tau \in \Sigma_{\text{Tor}}^{\mathcal{L}}} \text{Stop}(F)(\text{Run}(s, \alpha, \tau))$. If $\text{Val}_F^{\mathcal{L}}(s) = \overline{\text{Val}}_F^{\mathcal{L}}(s)$ then we say that the reachability game is determined, or the value $\text{Val}_F^{\mathcal{L}}(s)$ of the reachability game exists and it is such that $\text{Val}_F^{\mathcal{L}}(s) = \underline{\text{Val}}_F^{\mathcal{L}}(s) = \overline{\text{Val}}_F^{\mathcal{L}}(s)$. We say that Achilles wins the reachability game if $\text{Val}_F^{\mathcal{L}}(s) < \infty$. A *reachability game problem* is to decide whether in a given game arena G , an initial state s and a set of final states F , Achilles has a strategy to win the reachability game.

2.2 Reachability Games on Recursive state machines

A *recursive state machine* [2] \mathcal{M} is a tuple $(\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_k)$ of components, where each component $\mathcal{M}_i = (N_i, \text{EN}_i, \text{EX}_i, B_i, Y_i, A_i, X_i)$ for each $1 \leq i \leq k$ is such that:

- N_i is a finite set of *nodes* including a distinguished set EN_i of *entry nodes* and a set EX_i of *exit nodes* such that EX_i and EN_i are disjoint sets;
- B_i is a finite set of *boxes*;
- $Y_i : B_i \rightarrow \{1, 2, \dots, k\}$ is a mapping that assigns every box to a component. We associate a set of *call ports* $\text{Call}(b)$ and return ports $\text{Ret}(b)$ to each box $b \in B_i$:

$$\text{Call}(b) = \{(b, en) : en \in \text{EN}_{Y_i(b)}\} \text{ and } \text{Ret}(b) = \{(b, ex) : ex \in \text{EX}_{Y_i(b)}\}.$$

Let $\text{Call}_i = \cup_{b \in B_i} \text{Call}(b)$ and $\text{Ret}_i = \cup_{b \in B_i} \text{Ret}(b)$ be the set of call and return ports of component \mathcal{M}_i . We define the set of locations Q_i of component \mathcal{M}_i as the union of the set of nodes, call ports and return ports, i.e. $Q_i = N_i \cup \text{Call}_i \cup \text{Ret}_i$;

- A_i is a finite set of *actions*; and
- $X_i : Q_i \times A_i \rightarrow Q_i$ is the transition function with a condition that call ports and exit nodes do not have any outgoing transitions.

For the sake of simplicity, we assume that the set of boxes B_1, \dots, B_k and set of nodes N_1, N_2, \dots, N_k are mutually disjoint. We use symbols N, B, A, Q, X , etc. to denote the union of the corresponding symbols over all components.

An example of a RSM is shown in Figure 2 (taken from [20]). An execution of a RSM begins at the entry node of some component and depending upon the sequence

of input actions the state evolves naturally like a labeled transition system. However, when the execution reaches an entry port of a box, this box is stored on a stack of pending calls, and the execution continues naturally from the corresponding entry node of the component mapped to that box. When an exit node of a component is encountered, and if the stack of pending calls is empty then the run terminates; otherwise, it pops the box from the top of the stack and jumps to the exit port of the just popped box corresponding to the just reached exit of the component. We formalize the semantics of a RSM using a discrete LTS, whose states are pairs consisting of a sequence of boxes, called the context, mimicking the stack of pending calls and the current location. Let $\mathcal{M} = (\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_k)$ be an RSM where the component \mathcal{M}_i is $(N_i, En_i, Ex_i, B_i, Y_i, A_i, X_i)$. The semantics of \mathcal{M} is the discrete labelled transition system $\llbracket \mathcal{M} \rrbracket = (S_{\mathcal{M}}, A_{\mathcal{M}}, X_{\mathcal{M}})$ where:

- $S_{\mathcal{M}} \subseteq B^* \times Q$ is the set of states;
- $A_{\mathcal{M}} = \cup_{i=1}^k A_i$ is the set of actions;
- $X_{\mathcal{M}} : S_{\mathcal{M}} \times A_{\mathcal{M}} \rightarrow S_{\mathcal{M}}$ is the transition function such that for $s = (\langle \kappa \rangle, q) \in S_{\mathcal{M}}$ and $a \in A_{\mathcal{M}}$, we have that $s' = X_{\mathcal{M}}(s, a)$ if and only if one of the following holds:
 1. the location q is a call port, i.e. $q = (b, en) \in \text{Call}$, and $s' = (\langle \kappa, b \rangle, en)$;
 2. the location q is an exit node, i.e. $q = ex \in \text{EX}$ and $s' = (\langle \kappa' \rangle, (b, ex))$ where $(b, ex) \in \text{Ret}(b)$ and $\kappa = (\kappa', b)$;
 3. the location q is any other kind of location, and $s' = (\langle \kappa \rangle, q')$ and $q' \in X(q, a)$.

Given \mathcal{M} and a subset $Q' \subseteq Q$ of its nodes we define $\llbracket Q' \rrbracket_{\mathcal{M}}$ as $\{(\langle \kappa \rangle, v') : \kappa \in B^* \text{ and } v' \in Q'\}$. We define the terminal configurations $\text{Term}_{\mathcal{M}}$ as the set $\{(\langle \varepsilon \rangle, ex) : ex \in \text{EX}\}$ with the empty context $\langle \varepsilon \rangle$. Given a recursive state machine \mathcal{M} , an initial node v , and a set of final locations $F \subseteq Q$ the *reachability problem* on \mathcal{M} is defined as the reachability problem on the LTS $\llbracket \mathcal{M} \rrbracket$ with the initial state $(\langle \varepsilon \rangle, v)$ and final states $\llbracket F \rrbracket$. We define *termination problem* as the reachability of one of the exits with the empty context. The reachability and the termination problem for recursive state machines can be solved in polynomial time [2].

A partition $(Q_{\text{Ach}}, Q_{\text{Tor}})$ of locations Q of an RSM \mathcal{M} (between Achilles and Toroise) gives rise to recursive game arena $G = (\mathcal{M}, Q_{\text{Ach}}, Q_{\text{Tor}})$. Given an initial state, v , and a set of final states, F , the reachability game on \mathcal{M} is defined as the reachability game on the game arena $(\llbracket \mathcal{M} \rrbracket, \llbracket Q_{\text{Ach}} \rrbracket_{\mathcal{M}}, \llbracket Q_{\text{Tor}} \rrbracket_{\mathcal{M}})$ with the initial state $(\langle \varepsilon \rangle, v)$ and the set of final states $\llbracket F \rrbracket_{\mathcal{M}}$. Also, the termination game \mathcal{M} is defined as the reachability game on the game arena $(\llbracket \mathcal{M} \rrbracket, \llbracket Q_{\text{Ach}} \rrbracket_{\mathcal{M}}, \llbracket Q_{\text{Tor}} \rrbracket_{\mathcal{M}})$ with the initial state $(\langle \varepsilon \rangle, v)$ and the set of final states $\text{Term}_{\mathcal{M}}$. It is a well known result (see, e.g. [21], [14]) that reachability games and termination games on RSMs are determined and decidable (EXPTIME-complete).

3 Recursive Hybrid Automata

Recursive hybrid automata (RHAs) extend classical hybrid automata (HAs) with recursion in a similar way RSMs extend LTSs. We study a rather simpler subclass of HA known as singular hybrid automata where all variables grow with constant-rates.

3.1 Syntax

Let \mathbb{R} be the set of real numbers. Let \mathcal{X} be a finite set of real-valued variables. A *valuation* on \mathcal{X} is a function $\nu : \mathcal{X} \rightarrow \mathbb{R}$. We assume an arbitrary but fixed ordering on the variables and write x_i for the variable with order i . This allows us to treat a valuation ν as a point $(\nu(x_1), \nu(x_2), \dots, \nu(x_n)) \in \mathbb{R}^{|\mathcal{X}|}$. Abusing notations slightly, we use a valuation on \mathcal{X} and a point in $\mathbb{R}^{|\mathcal{X}|}$ interchangeably. For a subset of variables $X \subseteq \mathcal{X}$ and a valuation $\nu' \in \mathcal{X}$, we write $\nu[X:=\nu']$ for the valuation where $\nu[X:=\nu'](x) = \nu'(x)$ if $x \in X$, and $\nu[X:=\nu'](x) = \nu(x)$ otherwise. The valuation $\mathbf{0} \in \mathbb{R}^{|\mathcal{X}|}$ is a special valuation such that $\mathbf{0}(x) = 0$ for all $x \in \mathcal{X}$.

We define a constraint over a set \mathcal{X} as a subset of $\mathbb{R}^{|\mathcal{X}|}$. We say that a constraint is *rectangular* if it is defined as the conjunction of a finite set of constraints of the form $x \bowtie k$, where $k \in \mathbb{Z}$, $x \in \mathcal{X}$, and $\bowtie \in \{<, \leq, =, >, \geq\}$. For a constraint G , we write $\llbracket G \rrbracket$ for the set of valuations in $\mathbb{R}^{|\mathcal{X}|}$ satisfying the constraint G . We write \top (resp., \perp) for the special constraint that is true (resp., false) in all the valuations, i.e. $\llbracket \top \rrbracket = \mathbb{R}^{|\mathcal{X}|}$ (resp., $\llbracket \perp \rrbracket = \emptyset$). We write $\text{rect}(\mathcal{X})$ for the set of rectangular constraints over \mathcal{X} including \top and \perp .

Definition 1 (Recursive Hybrid Automata). A recursive hybrid automaton $\mathcal{H} = (\mathcal{X}, (\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_k))$ is a pair made of a set of variables \mathcal{X} and a collection of components $(\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_k)$ where every component $\mathcal{H}_i = (N_i, \text{EN}_i, \text{EX}_i, B_i, Y_i, A_i, X_i, P_i, \text{Inv}_i, E_i, J_i, F_i)$ is such that:

- N_i is a finite set of nodes including a distinguished set EN_i of entry nodes and a set EX_i of exit nodes such that EX_i and EN_i are disjoint sets;
- B_i is a finite set of boxes;
- $Y_i : B_i \rightarrow \{1, 2, \dots, k\}$ is a mapping that assigns every box to a component. (Call ports $\text{Call}(b)$ and return ports $\text{Ret}(b)$ of a box $b \in B_i$, and call ports Call_i and return ports Ret_i of a component \mathcal{H}_i are defined as before. We set $Q_i = N_i \cup \text{Call}_i \cup \text{Ret}_i$ and refer to this set as the set of locations of \mathcal{H}_i .)
- A_i is a finite set of actions.
- $X_i : Q_i \times A_i \rightarrow Q_i$ is the transition function with a condition that call ports and exit nodes do not have any outgoing transitions.
- $P_i : B_i \rightarrow 2^{\mathcal{X}}$ is pass-by-value mapping that assigns every box the set of variables that are passed by value to the component mapped to the box; (The rest of the variables are assumed to be passed by reference.)
- $\text{Inv}_i : Q_i \rightarrow \text{rect}(\mathcal{X})$ is the invariant condition;
- $E_i : Q_i \times A_i \rightarrow \text{rect}(\mathcal{X})$ is the action enabledness function;
- $J_i : A_i \rightarrow 2^{\mathcal{X}}$ is the variable reset function; and
- $F_i : Q_i \rightarrow \mathbb{N}^{|\mathcal{X}|}$ is the flow function characterizing the rate of each variable in each location.

We assume that the sets of boxes, nodes, locations, etc. are mutually disjoint across components and we write $(N, B, Y, Q, P, X, \text{etc.})$ to denote corresponding union over all components.

We say that a recursive hybrid automaton is *glitch-free* if for every box either all variables are passed by value or none is passed by value, i.e. for each $b \in B$ we have that

either $P(b) = \mathcal{X}$ or $P(b) = \emptyset$. Any general recursive hybrid automaton with one variable is trivially glitch-free. We say that a RHA is *hierarchical* if there exists an ordering over components such that a component never invokes another component of higher order or same order.

We say that a variable $x \in \mathcal{X}$ is a *clock* (resp., a stopwatch) if for every location $q \in Q$ we have that $F(q)(x) = 1$ (resp., $F(q)(x) \in \{0, 1\}$). A recursive timed automaton (RTA) is simply a recursive hybrid automata where all variables $x \in \mathcal{X}$ are clocks. Similarly, we define a recursive stopwatch automaton (RSA) as a recursive hybrid automaton where all variables $x \in \mathcal{X}$ are stopwatches. Since all of our results pertaining to recursive hybrid automata are shown in the context of recursive stopwatch automata, we often confuse RHA with RSA.

3.2 Semantics

A *configuration* of an RHA \mathcal{H} is a tuple $(\langle \kappa \rangle, q, \nu)$, where $\kappa \in (B \times \mathbb{R}^{|\mathcal{X}|})^*$ is sequence of pairs of boxes and variable valuations, $q \in Q$ is a location and $\nu \in \mathbb{R}^{|\mathcal{X}|}$ is a variable valuation over \mathcal{X} such that $\nu \in \text{Inv}(q)$. The sequence $\langle \kappa \rangle \in (B \times \mathbb{R}^{|\mathcal{X}|})^*$ denotes the stack of pending recursive calls and the valuation of all the variables at the moment that call was made, and we refer to this sequence as the context of the configuration. Technically, it suffices to store the valuation of variables passed by value, because other variables retain their value after returning from a call to a box, but storing all of them simplifies the notation. We denote the empty context by $\langle \epsilon \rangle$. For any $t \in \mathbb{R}$, we let $(\langle \kappa \rangle, q, \nu) + t$ equal the configuration $(\langle \kappa \rangle, q, \nu + F(q) \cdot t)$. Informally, the behaviour of an RHA is as follows. In configuration $(\langle \kappa \rangle, q, \nu)$ time passes before an available action is triggered, after which a discrete transition occurs. Time passage is available only if the invariant condition $\text{Inv}(q)$ is satisfied while time elapses, and an action a can be chosen after time t elapses only if it is enabled after time elapse, i.e., if $\nu + F(q) \cdot t \in E(q, a)$. If the action a is chosen then the successor state is $(\langle \kappa \rangle, q', \nu')$ where $q' \in X(q, a)$ and $\nu' = (\nu + t)[J(a) := \mathbf{0}]$. Formally, the semantics of an RHA is given by an LTS which has both an uncountably infinite number of states and transitions.

Definition 2 (RHA semantics). Let $\mathcal{H} = (\mathcal{X}, (\mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_k))$ be an RHA where each component is of the form $\mathcal{H}_i = (N_i, \text{EN}_i, \text{EX}_i, B_i, Y_i, A_i, X_i, P_i, \text{Inv}_i, E_i, J_i, F_i)$. The semantics of \mathcal{H} is a labelled transition system $\llbracket \mathcal{H} \rrbracket = (S_{\mathcal{H}}, A_{\mathcal{H}}, X_{\mathcal{H}})$ where:

- $S_{\mathcal{H}} \subseteq (B \times \mathbb{R}^{|\mathcal{X}|})^* \times Q \times \mathbb{R}^{|\mathcal{X}|}$, the set of states, is s.t. $(\langle \kappa \rangle, q, \nu) \in S_{\mathcal{H}}$ if $\nu \in \text{Inv}(q)$.
- $A_{\mathcal{H}} = \mathbb{R}_{\oplus} \times A$ is the set of timed actions, where \mathbb{R}_{\oplus} is the set of non-negative reals;
- $X_{\mathcal{H}} : S_{\mathcal{H}} \times A_{\mathcal{H}} \rightarrow S_{\mathcal{H}}$ is the transition function such that for $(\langle \kappa \rangle, q, \nu) \in S_{\mathcal{H}}$ and $(t, a) \in A_{\mathcal{H}}$, we have $(\langle \kappa' \rangle, q', \nu') = X_{\mathcal{H}}((\langle \kappa \rangle, q, \nu), (t, a))$ if and only if the following condition holds:
 1. if the location q is a call port, i.e. $q = (b, \text{en}) \in \text{Call}$ then $t = 0$, the context $\langle \kappa' \rangle = \langle \kappa, (b, \nu) \rangle$, $q' = \text{en}$, and $\nu' = \nu$.
 2. if the location q is an exit node, i.e. $q = \text{ex} \in \text{Ex}$, $\langle \kappa \rangle = \langle \kappa'', (b, \nu'') \rangle$, and let $(b, \text{ex}) \in \text{Ret}(b)$, then $t = 0$; $\langle \kappa' \rangle = \langle \kappa'' \rangle$; $q' = (b, \text{ex})$; and $\nu' = \nu[P(b) := \nu'']$.
 3. if location q is any other kind of location, then $\langle \kappa' \rangle = \langle \kappa \rangle$, $q' \in X(q, a)$, and
 - (a) $\nu + F(q) \cdot t' \in \text{Inv}(q)$ for all $t' \in [0, t]$;
 - (b) $\nu + F(q) \cdot t \in E(q, a)$;
 - (c) $\nu' = (\nu + F(q) \cdot t)[J(a) := \mathbf{0}]$.

3.3 Reachability and Time-Bounded Reachability Game Problems

For a subset $Q' \subseteq Q$ of states of RHA \mathcal{H} we define the set $\llbracket Q' \rrbracket_{\mathcal{H}}$ as the set $\{(\langle \kappa \rangle, q, \nu) \in S_{\mathcal{H}} : q \in Q'\}$. We define the terminal configurations as $Term_{\mathcal{H}} = \{(\langle \varepsilon \rangle, q, \nu) \in S_{\mathcal{H}} : q \in EX\}$. Given a recursive hybrid automaton \mathcal{H} , an initial node q and valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$, and a set of *final locations* $F \subseteq Q$, the *reachability problem* on \mathcal{H} is to decide the existence of a run in the LTS $\llbracket \mathcal{H} \rrbracket$ starting from the initial state $(\langle \varepsilon \rangle, q, \nu)$ to some state in $\llbracket F \rrbracket_{\mathcal{H}}$. As with RSMs, we also define *termination problem* as reachability of one of the exits with the empty context. Hence, given an RHA \mathcal{H} and an initial node q and a valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$, the termination problem on \mathcal{H} is to decide the existence of a run in the LTS $\llbracket \mathcal{H} \rrbracket$ from initial state $(\langle \varepsilon \rangle, q, \nu)$ to a final state in $Term_{\mathcal{H}}$.

Given a run $r = \langle s_0, (t_1, a_1), s_2, (t_2, a_2), \dots, (s_n, t_n) \rangle$ of an RHA, its time duration $time(r)$ is defined as $\sum_{i=1}^n t_i$. Given a recursive hybrid automaton \mathcal{H} , an initial node q , a bound $T \in \mathbb{N}$, and valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$, and a set of *final locations* $F \subseteq Q$, the *time-bounded reachability problem* on \mathcal{H} is to decide the existence of a run r in the LTS $\llbracket \mathcal{H} \rrbracket$ starting from the initial state $(\langle \varepsilon \rangle, q, \nu)$ to some state in $\llbracket F \rrbracket_{\mathcal{H}}$ such that $time(r) \leq T$. Time-bounded termination problem is defined in an analogous manner.

A partition (Q_{Ach}, Q_{Tor}) of locations Q of an RHA \mathcal{H} gives rise to a recursive hybrid game arena $\Gamma = (\mathcal{H}, Q_{Ach}, Q_{Tor})$. Given an initial location q , a valuation $\nu \in V$ and a set of final states F , the reachability game on Γ is defined as the reachability game on the game arena $(\llbracket \mathcal{H} \rrbracket, \llbracket Q_{Ach} \rrbracket_{\mathcal{T}}, \llbracket Q_{Tor} \rrbracket_{\mathcal{T}})$ with the initial state $(\langle \varepsilon \rangle, (q, \nu))$ and the set of final states $\llbracket F \rrbracket_{\mathcal{T}}$. Also, termination game on \mathcal{T} is defined as the reachability game on the game arena $(\llbracket \mathcal{T} \rrbracket, \llbracket Q_{Ach} \rrbracket_{\mathcal{T}}, \llbracket Q_{Tor} \rrbracket_{\mathcal{T}})$ with the initial state $(\langle \varepsilon \rangle, (q, \nu))$ and the set of final states $Term_{\mathcal{T}}$.

We prove the following key theorem about reachability games on various subclasses of recursive hybrid automata in Section 5.

Theorem 1. *The reachability game problem is undecidable for:*

1. *Unrestricted RSA with 2 stopwatches,*
2. *Glitchfree RSA with 3 stopwatches,*
3. *Unrestricted RTA with 3 clocks under bounded time, and*
4. *Glitchfree RSA with 4 stopwatches under bounded time.*

Moreover, all of these results hold even under hierarchical restriction.

On a positive side, we observe that for glitch-free RSA with two stopwatches reachability games are decidable by exploiting the existence of finite bisimulation for hybrid automata with 2 stopwatches. Details are given in Appendix 7.1

Theorem 2. *The reachability games are decidable for glitch-free RSA with atmost two stopwatches.*

We study the above mentioned problems when studied for a single player game. These problems have been detailed in Section 4.

Theorem 3. *The reachability problem is undecidable for*

1. *Unrestricted RHA with 2 stopwatches,*

2. *Glitchfree RHA with 3 stopwatches,*
3. *Unrestricted RTA with 5 clocks under bounded time, and*
4. *Glitchfree RHA with 14 stopwatches under bounded time.*

Moreover, all of these results hold even under hierarchical restriction.

On a positive side, we observe the following decidability results.

Theorem 4. *The reachability and the termination problems are decidable for*

1. *Glitch-free RHA with atmost two stopwatches*
2. *Bounded context RHA under bounded time, where variables are always passed-by-reference.*

The result for Glitch-free RHA with two stopwatches follows from the decidability of two stopwatch hybrid automata.

4 Undecidability Results with one player

In this section, we provide a proof sketch of our undecidability results by reducing the halting problem for two counter machines to the reachability problem in an RHA/RTA. A *two-counter machine* M is a tuple (L, C) where $L = \{\ell_0, \ell_1, \dots, \ell_n\}$ is the set of instructions including a distinguished terminal instruction ℓ_n called HALT, and the set $C = \{c_1, c_2\}$ of two *counters*. The instructions L are of the type:

1. (increment c) $\ell_i : c := c + 1$; goto ℓ_k ,
2. (decrement c) $\ell_i : c := c - 1$; goto ℓ_k ,
3. (zero-check c) $\ell_i : \text{if } (c > 0) \text{ then goto } \ell_k \text{ else goto } \ell_m$,

where $c \in C$, $\ell_i, \ell_k, \ell_m \in L$. A configuration of a two-counter machine is a tuple (l, c, d) where $l \in L$ is an instruction, and $c, d \in \mathbb{N}$ is the value of counters c_1 and c_2 , resp. A run of a two-counter machine is a (finite or infinite) sequence of configurations $\langle k_0, k_1, \dots \rangle$ where $k_0 = (\ell_0, 0, 0)$ and the relation between subsequent configurations is governed by transitions between respective instructions. The *halting problem* for a two-counter machine asks whether its unique run ends at the terminal instruction ℓ_n . It is well known ([18]) that the halting problem for two-counter machines is undecidable.

In order to prove four results of Theorem 3, we construct a recursive (timed/hybrid) automaton whose main components simulate various instructions. In these constructions the reachability of the exit node of each component corresponding to an instruction is linked to a faithful simulation of various increment, decrement and zero check instructions of the machine by choosing appropriate delays to adjust the clocks/variables, to reflect changes in counter values. We specify a main component for each instruction of the two counter machine. The entry node and exit node of a main component corresponding to an instruction $\ell_i : c := c + 1$; goto ℓ_k are respectively ℓ_i and ℓ_k . Similarly, a main component corresponding to a zero check instruction $\ell_i : \text{if } (c > 0) \text{ then goto } \ell_k \text{ else goto } \ell_m$, has a unique entry node ℓ_i , and two exit nodes corresponding to ℓ_k and ℓ_m respectively. We get the complete RHA for the two-counter machines when we connect these main components in the same sequence as the corresponding machine.

The halting problem of the two counter machine now reduces to the reachability (or termination) of an exit (HALT) node ℓ_n in some component.

For the correctness proofs, we represent runs in the RSA using three different forms of transitions $s \xrightarrow[t]{g,J} s'$, $s \rightsquigarrow s'$ and $s \xrightarrow[M(V)]{*} s'$ defined in the following way:

1. The transitions of the form $s \xrightarrow[t]{g,J} s'$, where $s = (\langle \kappa \rangle, n, \nu)$, $s' = (\langle \kappa \rangle, n', \nu')$ are configurations of the RHA, g is a constraint or guard on variables that enables the transition, J is a set of variables, and t is a real number, holds if there is a transition in the RHA from vertex n to n' with guard g and reset set J . Also, $\nu' = \nu + rt[J := 0]$, where r is the rate vector of state s .
2. The transitions of the form $s \rightsquigarrow s'$ where $s = (\langle \kappa \rangle, n, \nu)$, $s' = (\langle \kappa' \rangle, n', \nu')$ correspond to the following cases:
 - transitions from a call port to an entry node. That is, $n = (b, en)$ for some box $b \in B$ and $\kappa' = \langle \kappa, (b, \nu) \rangle$ and $n' = en \in \text{EN}$ while $\nu' = \nu$.
 - transitions from an exit node to a return port which restores values of the variables passed by value, that is, $\langle \kappa \rangle = \langle \kappa'', (b, \nu'') \rangle$, $n = ex \in \text{EX}$ and $n' = (b, ex) \in \text{Ret}(b)$ and $\kappa' = \kappa''$, while $\nu' = \nu[P(b) := \nu'']$.
3. The transitions of the form $s \xrightarrow[M(V)]{t} s'$, called summary edges, where $s = (\langle \kappa \rangle, n, \nu)$, $s' = (\langle \kappa \rangle, n', \nu')$ are such that $n = (b, en)$ and $n' = (b, ex)$ are call and return ports, respectively, of a box b mapped to M which passes by value to M , the variables in V . t is the time elapsed between the occurrences of (b, en) and (b, ex) . In other words, t is the time elapsed in the component M .

A configuration $(\langle \kappa \rangle, n, \nu)$ is also written as $(\langle \kappa \rangle, n, (\nu(x), \nu(y)))$.

4.1 Unrestricted RHA with 2 stopwatches

For all the four undecidability results, we construct a recursive automaton (timed/hybrid) as per the case, whose main components are the modules for the instructions and the counters are encoded in the variables of the automaton. In these reductions, the reachability of the exit node of each component corresponding to an instruction is linked to a faithful simulation of various increment, decrement and zero check instructions of the machine by choosing appropriate delays to adjust the clocks/variables, to reflect changes in counter values. We specify a main component for each type instruction of the two counter machine, for example \mathcal{H}_{inc} for increment. The entry node and exit node of a main component \mathcal{H}_{inc} corresponding to an instruction $[\ell_i : c := c + 1; \text{goto } \ell_k]$ are respectively ℓ_i and ℓ_k . Similarly, a main component corresponding to a zero check instruction $[\ell_i : \text{if } (c > 0) \text{ then goto } \ell_k \text{ else goto } \ell_m]$, has a unique entry node ℓ_i , and two exit nodes corresponding to ℓ_k and ℓ_m respectively. The various main components corresponding to the various instructions, when connected appropriately, gives the higher level component \mathcal{H}_M and this completes the RHA \mathcal{H} . The entry node of \mathcal{H}_M is the entry node of the main component for the first instruction of M and the exit node is $Halt$. Suppose each main component for each type of instruction correctly simulates the instruction by accurately updating the counters encoded in the variables of \mathcal{H} . Then,

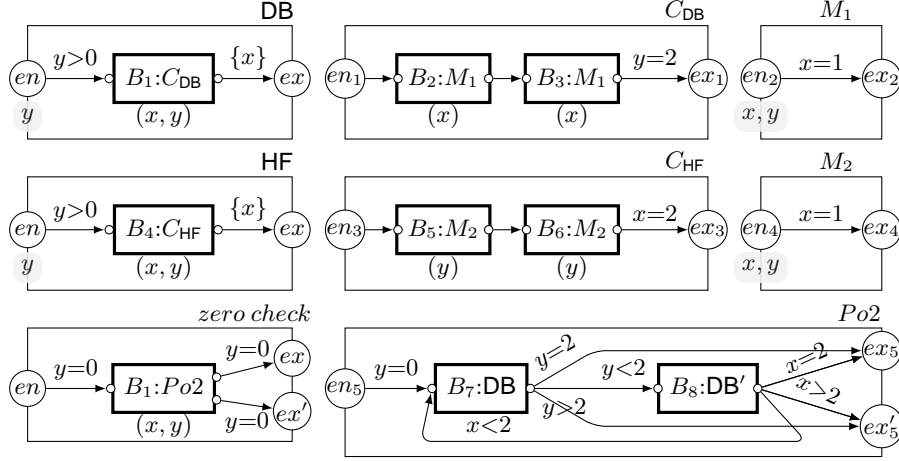


Fig. 3. RHA : 2 stopwatch : Decrement c is DB, increment c is HF and zero check d

the unique run in M corresponds to an unique run in \mathcal{H}_M . The halting problem of the two counter machine now boils down to the reachability of an exit node $Halt$ in \mathcal{H}_M .

Lemma 1. *The reachability problem is undecidable for recursive hybrid automata with at least two stopwatches.*

Proof. We prove that the reachability and termination problems are undecidable for 2 stopwatch unrestricted RHA. In order to obtain the undecidability result, we use a reduction from the halting problem for two counter machines. Our reduction uses a RHA with stopwatches x, y .

We specify a main component for each instruction of the two counter machine. On entry into a main component for increment/decrement/zero check, we have $x = \frac{1}{2^{c_3d}}$, $y = 0$ or $x = 0, y = \frac{1}{2^{c_3d}}$, where c, d are the current values of the counters. Given a two counter machine, we build a 2 stopwatch RHA whose building blocks are the main components for the instructions. The purpose of the components is to simulate faithfully the counter machine by choosing appropriate delays to adjust the variables to reflect changes in counter values. On entering the entry node en of a main component corresponding to an instruction ℓ_i , we have the configuration $(\langle \epsilon \rangle, en, (\frac{1}{2^{c_3d}}, 0))$ or $(\langle \epsilon \rangle, en, (0, \frac{1}{2^{c_3d}}))$ of the two stopwatch RHA.

We shall now present the components for increment/decrement and zero check instructions. In all the components, the ticking variables are written below respective locations in grey, while the variables passed by value are written below the boxes.

Simulate decrement instruction: Lets consider the decrement instruction $\ell_i: c = c - 1; \text{ goto } \ell_k$. Figure 3 gives the component DB which decrements counter c , by doubling $\frac{1}{2^{c_3d}}$. Assume that $x = \frac{1}{2^{c_3d}}$ and $y = 0$ on entering DB . Lets denote by x_{old} the value $\frac{1}{2^{c_3d}}$. A non-deterministic amount of time t is spent at the entry node en of DB . This makes $x = x_{old}$ and $y = t$, at the call port of $B_1 : C_{DB}$. Both x, y are passed by value to C_{DB} .

At the entry node of C_{DB} , the rates of x, y are zero. At sometime, the call port of $B_2 : M_1$ is reached with $x = x_{old}$ and $y = t$. M_1 is called by passing x by value. At the entry node of M_1 , a time $1 - x_{old}$ is spent, obtaining $x = 1, y = t + 1 - x_{old}$. We return from the exit node of M_1 to the return port of $B_2 : M_1$, with $x = x_{old}, y = 1 + t - x_{old}$. The rates of x, y are both zero here. After some time, we are at the call port of $B_3 : M_1$. Here again, M_1 is called by passing x by value. Going through M_1 again gives us $x = 1, y = 2 - 2x_{old} + t$. At the return port of B_3 , we thus have $x = x_{old}, y = 2 - 2x_{old} + t$. Again since the rates of x, y are both zero at the return port of B_3 , to get to the exit node of C_{DB} , y must be exactly equal to 2. That is, $2x_{old} = t$. In that case, when we get back to the return port of $B_1 : C_{DB}$, we have $x = x_{old}, y = t$, with the guarantee that $t = 2x_{old}$. The rates of x, y are both zero here, so we get to the exit node ex of DB resetting x . Thus, when we reach ex , we have $x = 0, y = \frac{1}{2^{c-1}3^d}$.

Simulate increment instruction: The instruction $[\ell_i: c = c + 1; \text{goto } \ell_k]$ is handled by the component HF in Figure 3. The main component HF , when entered with $x = \frac{1}{2^{c-1}3^d}, y = 0$ will halve the value of x , and return $x = 0, y = \frac{1}{2^{c+1}3^d}$. The working of the component HF can be explained in a similar way as that of DB .

Zero check instruction: The component *zerocheck* simulating $[\ell_i : \text{if } (d > 0) \text{ then goto } \ell_k \text{ else goto } \ell_m]$ can be found in Figure 3. Assume we are at en with $x = \frac{1}{2^{c-1}3^d} = x_{old}$ and $y = 0$. The rates of both x, y are zero, so we reach the callport of $B_1:Po2$ with the same values of x, y . $Po2$ is called by passing both x, y by value. No time is spent at the entry node en_5 of $Po2$, so with $y = 0$, we reach the call port of $B_7:DB$. Recall that DB is the component that doubles the value of x and stores it in y ; DB is called by passing both x, y by reference; when we return to the return port of B_7 , we have $x = 0, y = 2x_{old}$. If $y = 2$, then we go straightaway to the exit node ex_5 of $Po2$. If $x < 2$, then we goto the callport of $B_8:DB'$ from the return port of B_7 . The component DB' is similar to DB , with the roles of x, y reversed as compared to DB , and entry to DB' happens with $x = 0, y = 2x_{old}$. At the exit node of DB' , we obtain $y = 0, x = 4x_{old}$. Now, if $x = 2$, then we goto the exit node ex_5 of $Po2$. If $x < 2$, then we goto the callport of $B_7:DB$. In this way, we alternate between DB, DB' until we have multiplied x_{old} by some number k such that $k.x_{old}$ is exactly 2. If we obtain $k.x_{old} = 2$ at the return port of B_7 , then we have $y = 2$ and $x = 0$, while if we obtain $k.x_{old} = 2$ at the return port of B_8 , then we have $x = 2$ and $y = 0$. If this happens, then $d = 0$. If $d > 0$, then we will never obtain $k.x_{old}$ as 2. In this case, we go to the exit node ex'_5 when x (or y) exceeds 2. If we reach the exit node ex_5 of $Po2$, then we goto the exit node ex of the zerocheck component, and if we reach the exit node ex'_5 of $Po2$, then we goto the exit node ex' of the zerocheck component.

The following propositions show the correctness of the increment, decrement and zero check components. For the correctness proofs, we represent runs in the RHA using three different forms of transitions $s \xrightarrow[t]{g, J} s', s \rightsquigarrow s'$ and $s \xrightarrow[M(V)]{*} s'$ defined in the following way:

1. The transitions of the form $s \xrightarrow[t]{g, J} s'$, where $s = (\langle \kappa \rangle, n, \nu)$, $s' = (\langle \kappa \rangle, n', \nu')$ are configurations of the RHA, g is a constraint or guard on variables that enables the transition, J is a set of variables, and t is a real number, holds if there is a

transition in the RHA from vertex n to n' with guard g and reset set J . Also, $\nu' = \nu + rt[J := 0]$, where r is the rate vector of state s .

2. The transitions of the form $s \rightsquigarrow s'$ where $s = (\langle \kappa \rangle, n, \nu)$, $s' = (\langle \kappa' \rangle, n', \nu')$ correspond to the following cases:
 - transitions from a call port to an entry node. That is, $n = (b, en)$ for some box $b \in B$ and $\kappa' = \langle \kappa, (b, \nu) \rangle$ and $n' = en \in \text{EN}$ while $\nu' = \nu$.
 - transitions from an exit node to a return port which restores values of the variables passed by value, that is, $\langle \kappa \rangle = \langle \kappa'', (b, \nu'') \rangle$, $n = ex \in \text{EX}$ and $n' = (b, ex) \in \text{Ret}(b)$ and $\kappa' = \kappa''$, while $\nu' = \nu[P(b) := \nu'']$.
3. The transitions of the form $s \xrightarrow[M(V)]{*} s'$, called summary edges, where $s = (\langle \kappa \rangle, n, \nu)$, $s' = (\langle \kappa' \rangle, n', \nu')$ are such that $n = (b, en)$ and $n' = (b, ex)$ are call and return ports, respectively, of a box b mapped to M which passes by value to M , the variables in V .

A configuration $(\langle \kappa \rangle, n, \nu)$ is also written as $(\langle \kappa \rangle, n, (\nu(x), \nu(y)))$.

Proposition 1. *For any context κ , any box $b \in B$, and $x \in [0, 1]$, we have that $(\langle \kappa \rangle, (b, en), (x, 0)) \xrightarrow[DB]{*} (\langle \kappa \rangle, (b, ex), (0, 2x))$*

Proof. Component DB uses components C_{DB} and M_1 . The following is a unique run starting from $(\langle \kappa \rangle, (b, en), (x, 0))$ terminating in $(\langle \kappa \rangle, (b, ex), (2x, 0))$.

$$(\langle \kappa \rangle, (b, en), (x_0, 0)) \rightsquigarrow (\langle \kappa, b \rangle, en, (x_0, 0))$$

$$\xrightarrow[t]{y>0, \emptyset} (\langle \kappa, b \rangle, (B_1, en_1), (x_0, t)) \rightsquigarrow (\langle \kappa, b, (B_1, (x_0, t)) \rangle, en_1, (x_0, t))$$

$$\xrightarrow[any]{true, \emptyset} (\langle \kappa, b, (B_1, (x_0, t)) \rangle, (B_2, en_2), (x_0, t)) \rightsquigarrow (\langle \kappa, b, (B_1, (x_0, t)) \rangle, (B_2, x_0), en_2, (x_0, t))$$

$$\xrightarrow[1-x_0]{x=1, \emptyset} (\langle \kappa, b, (B_1, (x_0, t)) \rangle, (B_2, x_0), ex_2, (1, 1 - x_0 + t))$$

$$\rightsquigarrow (\langle \kappa, b, (B_1, (x_0, t)) \rangle, (B_2, ex_2), (x_0, 1 - x_0 + t))$$

$$\xrightarrow[any]{true, \emptyset} (\langle \kappa, b, (B_1, (x_0, t)) \rangle, (B_3, en_2), (x_0, 1 - x_0 + t))$$

$$\rightsquigarrow (\langle \kappa, b, (B_1, (x_0, t)) \rangle, (B_3, x_0), en_2, (x_0, 1 - x_0 + t))$$

$$\xrightarrow[1-x_0]{x=1, \emptyset} (\langle \kappa, b, (B_1, (x_0, t)) \rangle, (B_3, x_0), ex_2, (1, 2 - 2x_0 + t))$$

$$\rightsquigarrow (\langle \kappa, b, (B_1, (x_0, t)) \rangle, (B_3, ex_2), (x_0, 2 - 2x_0 + t))$$

$$\xrightarrow[any]{y=2, \emptyset} (\langle \kappa, b, (B_1, (x_0, t)) \rangle, ex_1, (x_0, 2)) \quad (2 - 2x_0 + t = 2 \leftrightarrow t = 2x_0)$$

$$\rightsquigarrow (\langle \kappa, b \rangle, (B_1, ex_1), (x_0, 2x_0))$$

$$\xrightarrow[any]{true, \{x\}} (\langle \kappa, b \rangle, ex, (0, 2x_0)) \rightsquigarrow (\langle \kappa \rangle, (b, ex), (0, 2x_0)).$$

The transitions above easily follow from the descriptions given in the decrement section. \square

Proposition 2 proves the correctness of the component HF .

Proposition 2. *For any context κ , any box $b \in B$, and $x \in [0, 1]$, we have that $(\langle \kappa \rangle, (b, en), (x, 0)) \xrightarrow[HF]{*} (\langle \kappa \rangle, (b, ex), (0, \frac{x}{2}))$*

Proof. Similar to Proposition 1.

Proposition 3 proves the correctness of the component $Po2$ that checks if x is a power of 2:

Proposition 3. *For any context κ , any box $b \in B$, and $x \in [0, 1]$, we have that starting from $(\langle \kappa \rangle, (b, en), (x, 0))$, *zerocheck* terminates at $(\langle \kappa \rangle, (b, ex), (x, 0))$ iff $x = \frac{1}{2^i}$, $i \in \mathbb{N}$. Otherwise, it terminates in $(\langle \kappa \rangle, (b, ex'), (x, 0))$.*

Proof. The proof of this follows from the correctness of the component DB shown above. Indeed, if DB doubles the variable, clearly, $\frac{1}{2^{c3^d}}$ will become 2 eventually after $c + 1$ invocations of DB iff $d = 0$. \square

Note that the components for incrementing, decrementing and zero check for counter d can be obtained in a manner similar to DB, HF . The only difference is that we have to multiply and divide by 3; these gadgets can be obtained straightforwardly by adapting DB, HF appropriately.

We now show that the two counter machine halts iff a vertex *Halt* corresponding to the halting instruction is reached in the RHA. Clearly, all the main components discussed above ensure that all instructions are simulated correctly. Assume the two counter machine halts. Then clearly, after going through all the main components corresponding to relevant instructions, we reach the component that leads to the *Halt* vertex. The $DB, HF, zerocheck$ subcomponents again ensure that simulation is done correctly to reach the vertex *Halt*. Conversely, assume that the two counter machine does not halt. Then there are two possibilities: (1) the RHA proceeds component by component, forever, simulating all instructions faithfully, or (2) the RHA is unable to take a transition, due to an error in the simulation of instructions. In either case, the vertex *Halt* is never reached.

4.2 GlitchFree RHA with 3 stopwatches

Lemma 2. *The reachability problem is undecidable for recursive hybrid automata with at least three stopwatches.*

Proof. The proof of this Lemma is a straightforward adaptation of the techniques used in Lemma 4.1. The main difference here is that, at all times, we have to pass all variables either by value, or by reference. This necessitates the need for an extra variable. In particular, we always pass all variables only by value. Thus, our result holds for the case of “pass by value” RHAs with 3 stopwatches.

We specify a main component for each instruction of the two counter machine. On entry into a main component for increment/decrement/zero check, we have $x = \frac{1}{2^{c3^d}}$, $y = z = 0$ where c, d are the current values of the counters. Given a two counter machine, we build a 3 stopwatch RHA whose building blocks are the main components

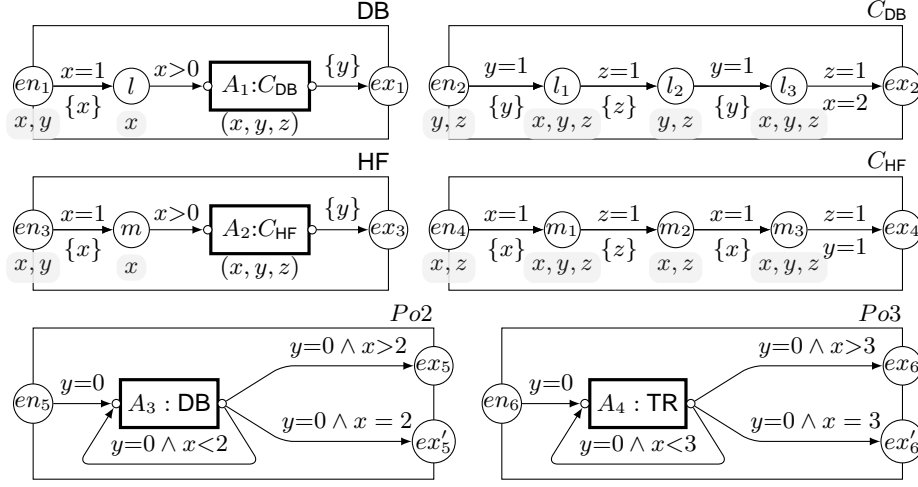


Fig. 4. Glitch-free RSA 3 stopwatch : Decrement c , Increment c and $Po2$, $Po3$

for the instructions. The purpose of the components is to simulate faithfully the counter machine by choosing appropriate delays to adjust the variables to reflect changes in counter values. On entering the entry node en of a main component corresponding to an instruction ℓ_i , we have the configuration $(\langle \epsilon \rangle, en, (\frac{1}{2^{c3^d}}, 0, 0))$ of the three stopwatch RHA. We shall now present the components for increment/decrement and zero check instructions. In all the components, the ticking variables are written below respective locations in grey.

Simulate decrement instruction: Lets consider the decrement instruction $\ell_i: c = c - 1; \text{ goto } \ell_k$. Figure 5 gives the component DB which decrements counter c , by doubling $\frac{1}{2^{c3^d}}$. Assume that $x = \frac{1}{2^{c3^d}}$ and $y = z = 0$ on entering DB . Lets denote by x_{old} the value $\frac{1}{2^{c3^d}}$. A time $1 - x_{old}$ is spent at the entry node en_1 of DB , resulting in $x = 0, y = 1 - x_{old}, z = 0$ at location l . A non-deterministic amount of time t is spent at l . This makes $x = t$ and $y = 1 - x_{old}, z = 0$, at the call port of $A_1 : C_{DB}$. All variables x, y, z are passed by value to C_{DB} .

At the entry node en_2 of C_{DB} , the rates of y, z are one. A time x_{old} is spent at en_2 , obtaining $y = 0, x = t$ and $z = x_{old}$ at l_1 . At l_1 , a time $1 - x_{old}$ is spent, obtaining $x = 1 + t - x_{old}, y = 1 - x_{old}$ and $z = 0$ at l_2 . A time x_{old} is spent at l_2 obtaining $x = 1 + t - x_{old}, y = 0, z = x_{old}$ at l_3 . A time $1 - x_{old}$ is spent at l_3 , obtaining $z = 1, x = 2 + t - 2x_{old}$ and $y = 1 - x_{old}$. To move out of l_3 , x must be 2; that is possible iff $t = 2x_{old}$. In this case, from the return port of $A_1 : C_{DB}$ (rates are all 0 here), we reach the exit node ex_1 of DB resetting y , obtaining $x = t = 2x_{old}, y = z = 0$, thereby successfully decrementing c .

Simulate increment instruction: The instruction $\ell_i: c = c + 1; \text{ goto } \ell_k$ is handled by the component HF in Figure 5. A time $1 - x_{old}$ is spent at entry node en_3 of HF , reaching location m with $x = 0, y = 1 - x_{old}$ and $z = 0$. A non-deterministic time

t is spent in m , reaching the entry node en_4 of C_{HF} with $x = t$, $y = 1 - x_{old}$ and $z = 0$. The exit node ex_4 of C_{HF} can be reached iff $t = \frac{x_{old}}{2}$. The working of these components are similar to DB, C_{DB} .

Zero check instruction: The component *zerocheck* simulating ℓ_i : if $(d > 0)$ then goto ℓ_k is the same as the *zerocheck* component in Figure 3, where the subcomponent *Po2* is called, passing all variables by value. The subcomponent *Po2* called can be found in Figure 5. At the entry node of *Po2*, no time is spent, and we are at the call port of *DB*.

We have drawn *DB* here like a box to avoid clutter, but it is actually a transition that goes from en_5 on $y = 0$ to a location called en_1 . Continue with the transitions drawn inside *DB* (treat them like normal transitions), and we have the sequence of transitions from en_1 to ex_1 , where C_{DB} is called in between. The edge $x < 2 \wedge y = 0$ is a transition from ex_1 to en_1 . In the figure, to avoid clutter, we have drawn it from the return port to the call port of *DB*. This loop from ex_1 to en_1 is invoked repeatedly, until we obtain x exactly equal to 2. If this happens, then we know that $d = 0$ in $\frac{1}{2c3^d} = x_{old}$. If this does not happen, then at some point of time, we will obtain x as more than 2. In this case, $d \neq 0$. In the former case, we go the exit node ex' of *Po2* from ex_1 , and in the latter case, we go to the exit node ex of *Po2* from ex_1 . Note that, whenever a box is called, we have always passed all the variables only by value.

The propositions proving correctness of the main and sub components is similar to Lemma 4.1. Also, it is clear that the node *Halt* is reached iff the two counter machine halts. \square

4.3 GlitchFree RHA with 2 clocks and 1 stopwatch

Lemma 3. *The reachability problem is undecidable for recursive hybrid automata with at least two clocks and one stopwatch.*

Proof. The proof of this Lemma is a straightforward adaptation of the techniques used in Lemma 4.1. The main difference here is that, at all times, we have to pass all variables either by value, or by reference. This necessitates the need for an extra variable. In particular, we pass all variables by reference in all except the zero check module. Note that all the calls with pass by reference can be removed by expanding the sub-component (callee) in the main component (caller). Thus, our result holds for the case of “pass by value” RHAs with 2 clocks and 1 stopwatch.

We specify a main component for each instruction of the two counter machine. On entry into a main component for increment/decrement/zero check, we have two clocks $x = \frac{1}{2c3^d}$, $y = 0$ and one stopwatch $s = 0$, where c, d are the current values of the counters. Given a two counter machine, we build a 3 stopwatch RHA whose building blocks are the main components for the instructions. The purpose of the components is to simulate faithfully the counter machine by choosing appropriate delays to adjust the variables to reflect changes in counter values. On entering the entry node en of a main component corresponding to an instruction ℓ_i , we have the configuration $(\langle \epsilon \rangle, en, (\frac{1}{2c3^d}, 0, 0))$ of the three stopwatch RHA. We shall now present the components for increment/decrement and zero check instructions. In all the components, the ticking variables are written below respective locations in grey.

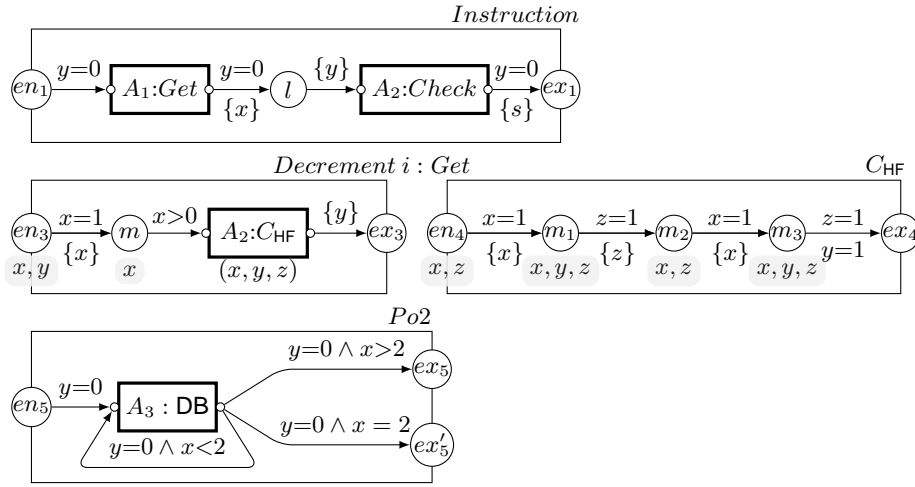


Fig. 5. Glitch-free RSA 3 stopwatch : Decrement c , Increment c and $Po2$, $Po3$

Simulate decrement instruction: Lets consider the decrement instruction $\ell_i: c = c - 1$; goto ℓ_k . Figure 5 gives the component DB which decrements counter c , by doubling $\frac{1}{2^{c_3d}}$. Assume that $x = \frac{1}{2^{c_3d}}$ and $y = z = 0$ on entering DB . Lets denote by x_{old} the value $\frac{1}{2^{c_3d}}$. A time $1 - x_{old}$ is spent at the entry node en_1 of DB , resulting in $x = 0, y = 1 - x_{old}, z = 0$ at location l . A non-deterministic amount of time t is spent at l . This makes $x = t$ and $y = 1 - x_{old}, z = 0$, at the call port of $A_1 : C_{DB}$. All variables x, y, z are passed by value to C_{DB} .

At the entry node en_2 of C_{DB} , the rates of y, z are one. A time x_{old} is spent at en_2 , obtaining $y = 0, x = t$ and $z = x_{old}$ at l_1 . At l_1 , a time $1 - x_{old}$ is spent, obtaining $x = 1 + t - x_{old}, y = 1 - x_{old}$ and $z = 0$ at l_2 . A time x_{old} is spent at l_2 obtaining $x = 1 + t - x_{old}, y = 0, z = x_{old}$ at l_3 . A time $1 - x_{old}$ is spent at l_3 , obtaining $z = 1, x = 2 + t - 2x_{old}$ and $y = 1 - x_{old}$. To move out of l_3 , x must be 2; that is possible iff $t = 2x_{old}$. In this case, from the return port of $A_1 : C_{DB}$ (rates are all 0 here), we reach the exit node ex_1 of DB resetting y , obtaining $x = t = 2x_{old}, y = z = 0$, thereby successfully decrementing c .

Simulate increment instruction: The instruction $\ell_i: c = c + 1$; goto ℓ_k is handled by the component HF in Figure 5. A time $1 - x_{old}$ is spent at entry node en_3 of HF , reaching location m with $x = 0, y = 1 - x_{old}$ and $z = 0$. A non-deterministic time t is spent in m , reaching the entry node en_4 of C_{HF} with $x = t, y = 1 - x_{old}$ and $z = 0$. The exit node ex_4 of C_{HF} can be reached iff $t = \frac{x_{old}}{2}$. The working of these components are similar to DB, C_{DB} .

Zero check instruction: The component *zerocheck* simulating ℓ_i : if $(d > 0)$ then goto ℓ_k is the same as the *zerocheck* component in Figure 3, where the subcomponent $Po2$ is called, passing all variables by value. The subcomponent $Po2$ called can be found in Figure 5. At the entry node of $Po2$, no time is spent, and we are at the call port of DB .

We have drawn DB here like a box to avoid clutter, but it is actually a transition that goes from en_5 on $y = 0$ to a location called en_1 . Continue with the transitions drawn inside DB (treat them like normal transitions), and we have the sequence of transitions from en_1 to ex_1 , where C_{DB} is called in between. The edge $x < 2 \wedge y = 0$ is a transition from ex_1 to en_1 . In the figure, to avoid clutter, we have drawn it from the return port to the call port of DB. This loop from ex_1 to en_1 is invoked repeatedly, until we obtain x exactly equal to 2. If this happens, then we know that $d = 0$ in $\frac{1}{2^c 3^d} = x_{old}$. If this does not happen, then at some point of time, we will obtain x as more than 2. In this case, $d \neq 0$. In the former case, we go the exit node ex' of $Po2$ from ex_1 , and in the latter case, we go to the exit node ex of $Po2$ from ex_1 . Note that, whenever a box is called, we have always passed all the variables only by value.

The propositions proving correctness of the main and sub components is similar to Lemma 4.1. Also, it is clear that the node $Halt$ is reached iff the two counter machine halts. \square

4.4 Unrestricted RTA over bounded time

Lemma 4. *The time bounded reachability problem is undecidable for recursive timed automata with at least 5 clocks.*

Proof. We prove that the problem of reaching a chosen vertex in an RTA within 18 units of total elapsed time is undecidable. In order to get the undecidability result, we use a reduction from the halting problem for two counter machines. Our reduction uses an RTA with atleast 5 clocks.

We specify a main component for each instruction of the two counter machine. We maintain 3 sets of clocks. The first set $X = \{x\}$ encodes correctly the current value of counter c ; the second set $Y = \{y\}$ encodes correctly the current value of counter d ; the third set $Z = \{z_1, z_2\}$ of 2 clocks helps in zero-check. An extra clock b is used to enforce urgency in some locations. b is zero at the entry nodes of all the main components. Let \mathcal{X} denote the set of all 5 clocks. The tuple of variables written below each box denotes variables passed by value.

To be precise, on entry into a main component simulating the $(k+1)$ th instruction, we have the values of z_1, z_2 as $1 - \frac{1}{2^k}$, the value of x as $1 - \frac{1}{2^{c+k}}$, and the value of y as $1 - \frac{1}{2^{d+k}}$, where c, d are the current values of the counters after simulating the first k instructions. We will denote this by saying that at the beginning of the $(k+1)$ th instruction, we have $\nu(Z) = 1 - \frac{1}{2^k}$, $\nu(x) = 1 - \frac{1}{2^{c+k}}$ and $\nu(y) = 1 - \frac{1}{2^{d+k}}$. If the $(k+1)$ th instruction ℓ_{k+1} is an increment counter c instruction, then after the simulation of ℓ_{k+1} , we need $\nu(Z) = 1 - \frac{1}{2^{k+1}}$, $\nu(x) = 1 - \frac{1}{2^{c+k+2}}$ and $\nu(y) = 1 - \frac{1}{2^{d+k+1}}$. Similarly, if ℓ_{k+1} is a decrement c instruction, then after the simulation of ℓ_{k+1} , we need $\nu(Z) = 1 - \frac{1}{2^{k+1}}$, $\nu(x) = 1 - \frac{1}{2^{c+k}}$ and $\nu(y) = 1 - \frac{1}{2^{d+k+1}}$. Likewise, if ℓ_{k+1} is a zero check instruction, then after the simulation of ℓ_{k+1} , we need $\nu(Z) = 1 - \frac{1}{2^{k+1}}$, $\nu(x) = 1 - \frac{1}{2^{c+k+1}}$ and $\nu(y) = 1 - \frac{1}{2^{d+k+1}}$.

Simulate Increment Instruction: Let us discuss the case of simulating an increment instruction for counter c . Assume that this is the $(k+1)$ th instruction. Figure 6 gives the figure for incrementing counter c . At the entry node en_1 of the component $Inc\ c$, we have $\nu(x) = 1 - \frac{1}{2^{c+k}}$, $\nu(y) = 1 - \frac{1}{2^{d+k}}$ and $\nu(Z) = 1 - \frac{1}{2^k}$, and $\nu(b) = 0$.

The component *Inc c* has three subcomponents sequentially lined up one after the other: Let $\beta = \frac{1}{2^k}$, $\beta_c = \frac{1}{2^{c+k}}$, and $\beta_d = \frac{1}{2^{d+k}}$.

1. The first subcomponent is Up_2^y . If Up_2^y is entered with $\nu(y) = 1 - \beta_d$, then on exit, we have $\nu(y) = 1 - \frac{\beta_d}{2}$. The values of X, Z are unchanged. Also, the total time elapsed in Up_2^y is $\leq \frac{5\beta}{2}$.
2. The next subcomponent is Up_4^x . If Up_4^x is entered with $\nu(x) = 1 - \beta_c$, then on exit, we have $\nu(x) = 1 - \frac{\beta_c}{4}$. The values of Z, Y are unchanged. Also, the total time elapsed in Up_4^x is $\leq \frac{11\beta}{4}$.
3. The next subcomponent is Up_2^Z updates the value of Z . If Up_2^Z is entered with $\nu(Z) = 1 - \beta$, then on exit, we have $\nu(Z) = 1 - \frac{\beta}{2}$. The values of X, Y are unchanged. Also, the total time elapsed in Up_2^Z is $\leq \frac{5\beta}{2}$.
4. Thus, at the end of the *Inc c*, we obtain $\nu(Z) = 1 - \frac{1}{2^{k+1}}$, $\nu(x) = 1 - \frac{1}{2^{c+k+2}}$, $\nu(y) = 1 - \frac{1}{2^{d+k+1}}$. Also, the total time elapsed in *Inc c* is $\leq [\frac{5}{2} + \frac{11}{4} + \frac{5}{2}]\beta < 8\beta$.

On calling Up_n^a , for $a \in \{x, y\}$, the clock a is passed by reference; likewise, on calling Up_n^Z , clocks in Z are passed by reference. Here, $n \in \{2, 4\}$. Next, we describe the structure of the components Up_n^a for $a \in \{x, y\}$. At the entry node en_2 of Up_n^a , we have the invariant $b = 0$. Thus, no time is elapsed in the entry node en_1 of *Inc c* also. Up_n^a is made up of subcomponents $D, C_{z_2}^{a=}, D$ and Chk_n^a lined sequentially. Let us discuss the details of Up_2^y , the others have similar functionality.

1. On entry into the first subcomponent $F_4:D$, we have $\nu(Z) = 1 - \beta$, $\nu(b) = 0$, $\nu(x) = 1 - \beta_c$, $\nu(y) = 1 - \beta_d$. D is called, and clock z_2 is passed by reference and the rest by value. A non-deterministic amount of time t_1 elapses at the entry node en_3 of D . Back at the return port of $F_4:D$, we have clock z_2 added by t_1 .
2. We are then at the entry node of the subcomponent $F_5:C_{z_2}^{y=}$ with values $\nu(z_2) = 1 - \beta + t_1$, and $\nu(z_1) = 1 - \beta$, $\nu(x) = 1 - \beta_c$, $\nu(y) = 1 - \beta_d$ and $\nu(b) = 0$. $C_{z_2}^{y=}$ is called by passing all clocks by value. The subcomponent $C_{z_2}^{y=}$ ensures that $t_1 = \beta - \beta_d$.
3. To ensure $t_1 = \beta - \beta_d$, at the entry node en_4 of $C_{z_2}^{y=}$, a time β_d elapses. This makes $y = 1$. If z_2 must be 1, then we need $1 - \beta + t_1 + \beta_d = 1$, or the time t_1 elapsed is $\beta - \beta_d$. That is, $C_{z_2}^{y=}$ ensures that z_2 has grown to be equal to y by calling $F_4:D$. Back at the return port of $F_5:C_{z_2}^{y=}$, we next enter the call port of $F_6:D$ with $\nu(z_2) = \nu(y) = 1 - \beta_d$ and $\nu(z_1) = 1 - \beta$. D is called by passing y by reference, and all others by value. A non-deterministic amount of time t_2 is elapsed in D . At the return port of $F_6:D$, we get $\nu(z_1) = 1 - \beta$, $\nu(z_2) = 1 - \beta_d$, and $\nu(y) = 1 - \beta_d + t_2$.
4. At the call port of $F_7:Chk_2^y$, we have the same values, since $b = 0$ has to be satisfied at the exit node ex_7 of Chk_2^y . That is, at the call port of $F_7:Chk_2^y$, we have $\nu(z_1) = 1 - \beta$, $\nu(z_2) = 1 - \beta_d$, and $\nu(y) = 1 - \beta_d + t_2$. F_7 calls Chk_2^y , and passes all clocks by value. Chk_2^y checks that $t_2 = \frac{\beta_d}{2}$.
5. At the entry port en_7 of Chk_2^y , no time elapses. Chk_2^y sequentially calls M twice, each time passing z_2 by reference, and all others by value. In the first invocation of M , we want y to reach 1; thus a time $\beta_d - t_2$ is spent at en_6 . This makes $z_2 = 1 - \beta_d + \beta_d - t_2 = 1 - t_2$. After the second invocation, we obtain $z_2 = 1 + \beta_d - 2t_2$.

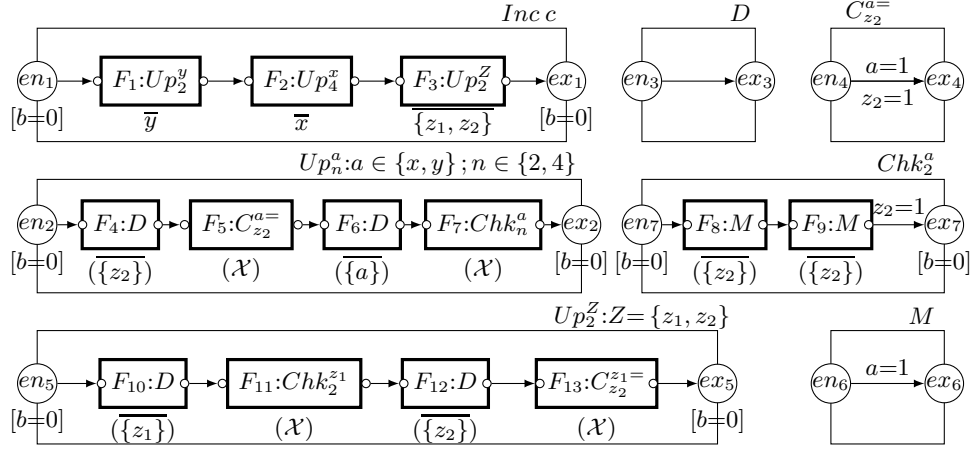


Fig. 6. *TB Term* in RTA: Increment c . Note that $\bar{S} = \mathcal{X} - S$ and $a \in \{x, y\}$. $C_{z_2}^{z_1=}$ is obtained by instantiating $a = z_1$ in $C_{z_2}^{a=}$. The component Chk_4^a is similar to Chk_2^a . It has 4 calls to M inside it each time passing only z_2 by reference. Zero Check component follows the same pattern as $Inc\ c$ calling Up_2^a , for all $a \in \{x, y\}$, followed by Up_2^Z , and then calls ZC passing all variables by value. ZC checks if $z_1 = x$ (with guard $z_1=1 \wedge x = 1$) to check if counter c is 0 and $z_1 = y$ to check if d is 0.

- at the return port of $F_9:M$. No time can elapse at the return port of $F_9:M$; for z_2 to be 1, we need $t_2 = \frac{\beta_d}{2}$.
- 6. No time elapses in the return port of $F_7:Chk_2^y$, and we are at the exit node ex_2 of Up_2^y .
- 7. Thus, at the exit node of Up_2^y , we have $\nu(z_1) = 1 - \beta$, $\nu(z_2) = 1 - \beta_d$ and $\nu(y) = 1 - \beta_d + t_2 = 1 - \frac{\beta_d}{2}$.
- 8. The time elapsed in Up_2^y is the sum of t_1, t_2 and the times elapsed in $C_{z_2}^{y=}$ and Chk_2^y . That is, $(\beta - \beta_d) + \frac{\beta_d}{2} + \beta_d + 2(\beta_d - t_2) = \beta + \frac{3\beta_d}{2} \leq \frac{5\beta}{2}$ since $\beta_d \leq \beta$.

At the return port of $F_1 : Up_2^y$, we thus have $\nu(Z) = 1 - \beta$ ($\nu(z)$ restored to $1 - \beta$ as it was passed by value to Up_2^y), $\nu(x) = 1 - \beta_c$, and $\nu(y) = 1 - \frac{\beta_d}{2}$. No time elapses here, and we are at the call port of $F_2 : Up_4^x$. The component Chk_4^x is similar to Chk_2^y . It has 4 calls to M inside it, each passing respectively, z_2 by reference to M and x by value. An analysis similar to the above gives that the total time elapsed in Up_4^x is $\leq \frac{11\beta}{4}$, and at the return port of $F_2 : Up_4^x$, we get $\nu(x) = 1 - \frac{\beta_c}{4}$, $\nu(y) = 1 - \frac{\beta_d}{2}$ and $\nu(Z) = 1 - \beta$. This is followed by entering $F_3 : Up_2^Z$, with these values. At the return port of $F_3 : Up_2^Z$, we obtain $\nu(x) = 1 - \frac{\beta_c}{4}$, $\nu(y) = 1 - \frac{\beta_d}{2}$ and $\nu(Z) = 1 - \frac{\beta}{2}$, with the total time elapsed in Up_2^Z being $\leq \frac{5\beta}{2}$.

From the explanations above, the following propositions can be proved. The same arguments given above will apply to prove this.

Proposition 4. For any box B and context $\langle \kappa \rangle$, and $\nu(Z) = 1 - \beta$, we have that $(\langle \kappa \rangle, (B, en), (\nu(x), \nu(y), 1 - \beta, \nu(b))) \xrightarrow{Up_2^Z} (\langle \kappa \rangle, (B, ex), (\nu(x), \nu(y), 1 - \frac{\beta}{2}, \nu(b)))$.

Proposition 5. For any box B and context $\langle \kappa \rangle$, and $\nu(x) = 1 - \beta_c$, we have that $(\langle \kappa \rangle, (B, en), (1 - \beta_c, \nu(y), \nu(Z), \nu(b))) \xrightarrow{Up_4^x} (\langle \kappa \rangle, (B, ex), (1 - \frac{\beta_c}{4}, \nu(y), \nu(Z), \nu(b)))$.

Proposition 6. For any box B and context $\langle \kappa \rangle$, and $\nu(y) = 1 - \beta_d$, we have that $(\langle \kappa \rangle, (B, en), (\nu(x), 1 - \beta_d, \nu(Z), \nu(b))) \xrightarrow{Up_2^y} (\langle \kappa \rangle, (B, ex), (\nu(x), 1 - \frac{\beta_d}{2}, \nu(Z), \nu(b)))$.

Simulate Decrement Instruction: Assume that the $(k + 1)$ st instruction is decrementing counter c . Then we construct the main component *Dec c* similar to the component *Inc c* above. The main change is the following:

- The main component *Dec c* will have the subcomponents Up_2^y and Up_2^Z lined up sequentially. There is no need for any Up_n^x subcomponent here, since the value of x stays unchanged on decrementing c . Also, the subcomponents Up_2^y and Up_2^Z do not alter the value of x : the functionality Up_2^Z and Up_2^y are the same as the one in *Inc c*. The total time spent in *Dec c* is also, less than 8β .

Zero Check Instruction: The main component for Zero Check follows the same pattern as *Inc c*. The main change is the following:

- The main component *ZeroCheck* will have the subcomponents Up_2^y and Up_2^x and Up_2^Z lined up sequentially. The functionality Up_2^Z , Up_2^x and Up_2^y are the same as the one in *Inc c*. After these three, we invoke a subcomponent *ZC*. *ZC* is called by passing all clocks by value. At the entry node *en* of *ZC*, we have two transitions, one on $z_1 = 1 \wedge x = 1$ leading to an exit node *ex*, and another one on $z_1 = 1 \wedge x \neq 1$ leading to *ex'*. Recall that $z_1 = \frac{1}{2^k}$. Thus, for z_1 to reach 1, a time elapse $\frac{\beta}{2} = \frac{1}{2^{k+1}}$ is needed. If this also makes $x = 1$, then we know that x on entry was $1 - \frac{1}{2^{c+k+1}} = 1 - \frac{1}{2^{k+1}}$ implying that $c = 0$. Likewise, if z_1 attains 1, but x does not, then $c \neq 0$. Since all clocks are passed by value, at the return port of *ZC* within the main component *ZeroCheck*, we regain back the clock values obtained after going through Up_2^Z , Up_2^x and Up_2^y : that is, $\nu(Z) = 1 - \frac{\beta}{2}$, $\nu(x) = 1 - \frac{\beta_c}{2}$ and $\nu(y) = 1 - \frac{\beta_d}{2}$. The time elapsed in *ZC* is $\frac{\beta}{2}$. The times elapsed in Up_2^Z and Up_2^x and Up_2^y are same as calculated in the case of Increment c . Thus, the total time elapsed here is $< 8\beta + \beta = 9\beta$.

We conclude by calculating the total time elapsed during the entire simulation. We have established so far that for the $(k + 1)$ th instruction, the time elapsed is no more than 9β , for $\beta = \frac{1}{2^k}$. For the first instruction, the time elapsed is at most 9, for the second instruction it is $\frac{9}{2}$, for the third it is $\frac{9}{2^2}$ and so on.

$$\text{Total time duration} = \begin{cases} 9(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots) \\ = 9(1 + (< 1)) \\ < 18 \end{cases} \quad (1)$$

Note that the components for incrementing, decrementing and zero check of counter d can be obtained in a manner similar to the above.

The proof that we reach the vertex *Halt* of the RTA iff the two counter machine halts follows: Clearly, the exit node of each main component is reached iff the corresponding instruction is simulated correctly. Thus, if the counter machine halts, we will indeed reach the exit node of the main component corresponding to the last instruction. However, if the machine does not halt, then we keep going between the various main components simulating each instruction, and never reach *Halt*. \square

4.5 Glitchfree RHA with 14 stopwatches

Lemma 5. *The time bounded reachability problem is undecidable for recursive hybrid automata with at least 14 stopwatches.*

Proof. We prove that the problem of reaching a chosen vertex in an RHA within 18 units of total elapsed time is undecidable. In order to get the undecidability result, we use a reduction from the halting problem for two counter machines. Our reduction uses an RHA with atleast 14 stopwatches.

We specify a main component for each instruction of the two counter machine. We maintain 3 sets of stopwatches. The first set $X = \{x_1, \dots, x_5\}$ encodes correctly the current value of counter c ; the second set $Y = \{y_1, \dots, y_5\}$ encodes correctly the current value of counter d ; and third set $Z = \{z_1, z_2, z_3\}$ encodes the end of (k) th instruction.

An extra stopwatch b is used to enforce urgency in some locations. b is zero at the entry nodes of all the main components. Let \mathcal{X} denote the set of all 14 stopwatches.

To be precise, on entry into a main component simulating the $(k+1)$ th instruction, we have the values of z_1, z_2, z_3 as $1 - \frac{1}{2^k}$, the values of x_1, \dots, x_5 as $1 - \frac{1}{2^{c+k}}$, and the values of y_1, \dots, y_5 as $1 - \frac{1}{2^{d+k}}$, where c, d are the current values of the counters after simulating the first k instructions. We will denote this by saying that at the beginning of the $(k+1)$ th instruction, we have $\nu(Z) = 1 - \frac{1}{2^k}$, $\nu(X) = 1 - \frac{1}{2^{c+k}}$ and $\nu(Y) = 1 - \frac{1}{2^{d+k}}$. If the $(k+1)$ th instruction ℓ_{k+1} is an increment counter c instruction, then after the simulation of ℓ_{k+1} , we need $\nu(Z) = 1 - \frac{1}{2^{k+1}}$, $\nu(X) = 1 - \frac{1}{2^{c+k+2}}$ and $\nu(Y) = 1 - \frac{1}{2^{d+k+1}}$. Similarly, if ℓ_{k+1} is a decrement c instruction, then after the simulation of ℓ_{k+1} , we need $\nu(Z) = 1 - \frac{1}{2^{k+1}}$, $\nu(X) = 1 - \frac{1}{2^{c+k}}$ and $\nu(Y) = 1 - \frac{1}{2^{d+k+1}}$. Likewise, if ℓ_{k+1} is a zero check instruction, then after the simulation of ℓ_{k+1} , we need $\nu(Z) = 1 - \frac{1}{2^{k+1}}$, $\nu(X) = 1 - \frac{1}{2^{c+k+1}}$ and $\nu(Y) = 1 - \frac{1}{2^{d+k+1}}$.

Simulate Increment Instruction: Let us discuss the case of simulating an increment instruction for counter c . Assume that this is the $(k+1)$ th instruction. Figure 7 gives the figure for incrementing counter c . At the entry node en_1 of the component *Inc c*, we have $\nu(X) = 1 - \frac{1}{2^{c+k}}$, $\nu(Y) = 1 - \frac{1}{2^{d+k}}$ and $\nu(Z) = 1 - \frac{1}{2^k}$, and $\nu(b) = 0$.

The component *Inc c* has three subcomponents sequentially lined up one after the other: Let $\beta = \frac{1}{2^k}$, $\beta_c = \frac{1}{2^{c+k}}$, and $\beta_d = \frac{1}{2^{d+k}}$.

1. The first subcomponent Up_2^Z (component UP_n^A in the figure 7 with A as Z and $n = 2$) updates the value of Z . If Up_2^Z is entered with $\nu(Z) = 1 - \beta$, then on exit, we have $\nu(Z) = 1 - \frac{\beta}{2}$. The values of X, Y are unchanged as their rate of growth is 0 throughout the component UP_2^Z and they are always passed by value to the subcomponents. Also, the total time elapsed in Up_2^Z is $\leq \frac{5\beta}{2}$.

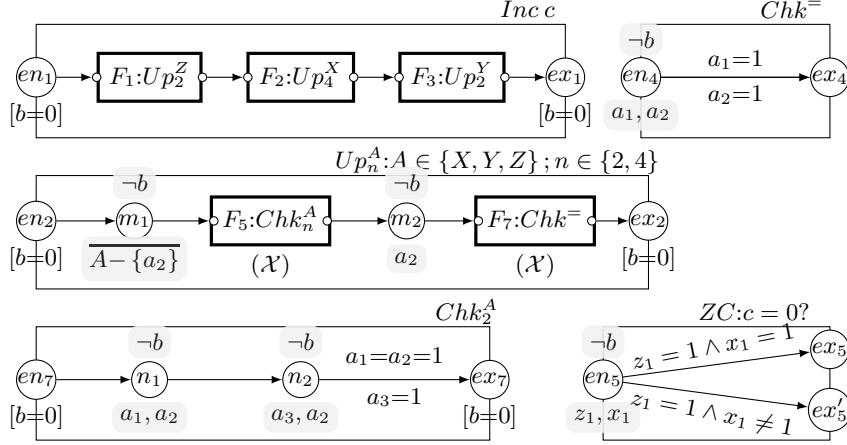


Fig. 7. Time bounded reachability in 14 stopwatch RSA: Increment c . Note that the variables which tick in a location are indicated below it. b ticks everywhere except in locations where it is specified as $\neg b$. Also \bar{S} denoted stopwatches $\mathcal{X} - S$.

2. The next subcomponent is Up_4^X . If Up_4^X is entered with $\nu(X) = 1 - \beta_c$, then on exit, we have $\nu(X) = 1 - \frac{\beta_c}{4}$. The values of Z, Y are unchanged. Also, the total time elapsed in Up_4^X is $\leq \frac{11\beta}{4}$.
3. The next subcomponent is Up_2^Y . If Up_2^Y is entered with $\nu(Y) = 1 - \beta_d$, then on exit, we have $\nu(Y) = 1 - \frac{\beta_d}{2}$. The values of X, Z are unchanged. Also, the total time elapsed in Up_2^Y is $\leq \frac{5\beta}{2}$.
4. Thus, at the end of the $Inc\ c$, we obtain $\nu(Z) = 1 - \frac{1}{2^{k+1}}$, $\nu(X) = 1 - \frac{1}{2^{c+k+2}}$, $\nu(Y) = 1 - \frac{1}{2^{d+k+1}}$. Also, the total time elapsed in $Inc\ c$ is $\leq [\frac{5}{2} + \frac{11}{4} + \frac{5}{2}]\beta < 8\beta$.

To avoid clutter, We have drawn Up_2^Z like a box inside $Inc\ c$; actually, think of it as the sequence of transitions from en_2 to ex_2 , with 2 boxes called in between. The same holds for “boxes” Up_4^X and Up_2^Y .

Next, we describe the structure of the components Up_n^A . At the entry node en_2 of Up_n^A , we have the invariant $b = 0$. Thus, no time is elapsed in the entry node en_1 of $Inc\ c$ also. Up_n^A is made up of subcomponents Chk_n^A and $Chk^=$. Let us discuss the details of Up_2^Z , the others have similar functionality.

1. On entry into the location m_1 , we have $\nu(Z) = 1 - \beta$, $\nu(b) = 0$, $\nu(X) = 1 - \beta_c$, $\nu(Y) = 1 - \beta_d$. In m_1 only stopwatches z_1, z_3 are grow. A non-deterministic amount of time t_1 elapses here. Thus when leaving m_1 , we have stopwatches z_1, z_3 added by t_1 .
2. We are then at the entry node of the subcomponent $F_5:Chk_2^Z$ with values $\nu(z_2) = 1 - \beta$, and $\nu(z_i) = 1 - \beta + t_1$ for $i = 1, 3$, $\nu(X) = 1 - \beta_c$, $\nu(Y) = 1 - \beta_d$ and $\nu(b) = 0$. Chk_2^Z is called by passing all stopwatches by value. The subcomponent Chk_2^Z ensures that $t_1 = \frac{\beta}{2}$.

3. To ensure $t_1 = \frac{\beta}{2}$, at the entry node en_7 of Chk_2^Z , no time can elapse. If t_2 and t_3 are times elapsed in n_1 and n_2 then, upon reaching exit node en_7 , we have $z_1 = 1 - \beta + t_1 + t_2$, $z_2 = 1 - \beta + t_2 + t_3$ and $z_3 = 1 - \beta + t_1 + t_3$. Additionally, $z_1 = z_2 = z_3 = 1$ implies $t_1 + t_2 = \beta = t_2 + t_3 = t_1 + t_3$. Thus, we get $t_1 = t_2 = t_3 = \frac{\beta}{2}$. Thus, the total time spent in Chk_2^Z is $t_2 + t_3 = \beta$.
4. At the return port of $F_5:Chk_2^Z$, we restore all values to what they were, at the call port of $F_5:Chk_2^Z$. That is, $\nu(z_2) = 1 - \beta$, and $\nu(z_i) = 1 - \beta + t_1$ for $i = 1, 3$, with the guarantee that $t_1 = \frac{\beta}{2}$.
A time t_4 is elapsed in location m_2 affecting only z_2 to become $z_2 = 1 - \beta + t_4$.
5. Finally, we call the subcomponent Chk^- with $z_1 = 1 - \frac{\beta}{2}$ and $z_2 = 1 - \beta + t_4$. All stopwatches are passed by value. Chk^- ensures that $z_1 = z_2$; that is, $t_4 = \frac{\beta}{2}$. A time $t_5 = \frac{\beta}{2}$ is spent at the entry node en_4 of Chk^- to ensure this. Thus, at the return port of $F_7:Chk^-$, we have $z_1 = z_2 = z_3 = 1 - \frac{\beta}{2}$, and the rest of the stopwatches unchanged. No time can elapse at the exit node ex_2 of Up_2^Z . Thus, at the return port of $F_1 : Up_2^Z$, we get $\nu(Z) = 1 - \frac{1}{2^{k+1}}$.
6. The total time elapsed in $F_1 : Up_2^Z$ is $t_1 + t_2 + t_3 + t_4 + t_5 = \frac{\beta}{2} + \beta + \frac{\beta}{2} + \frac{\beta}{2} = \frac{5\beta}{2}$.

At the return port of $F_2 : Up_2^Z$, we thus have $\nu(Z) = 1 - \frac{\beta}{2}$, $\nu(X) = 1 - \beta_c$, and $\nu(Y) = 1 - \beta_d$. No time elapses here, and we are at the call port of $F_2 : Up_4^X$. The component Chk_4^X is similar to Chk_2^Z . It has 4 locations h_1, h_3, h_4, h_5 inside it, each h_i has stopwatches x_2 and x_i ticking. An analysis similar to the above gives that the total time elapsed in Up_4^X is $\frac{11\beta}{4}$, and at the return port of $F_2 : Up_4^X$, we get $\nu(X) = 1 - \frac{\beta_c}{4}$, $\nu(Y) = 1 - \beta_d$ and $\nu(Z) = 1 - \frac{\beta}{2}$. This is followed by entering $F_3 : Up_2^Y$, with these values. At the return port of $F_3 : Up_2^Y$, we obtain $\nu(X) = 1 - \frac{\beta_c}{4}$, $\nu(Y) = 1 - \frac{\beta_d}{2}$ and $\nu(Z) = 1 - \frac{\beta}{2}$, with the total time elapsed in Up_2^Y being $\frac{5\beta_d}{4}$.

From the explanations above, the following propositions can be proved. The same arguments given above will apply to prove this.

Proposition 7. For any box B and context $\langle \kappa \rangle$, and $\nu(Z) = 1 - \beta$, we have that $((\kappa), (B, en), (\nu(X), \nu(Y), 1 - \beta, \nu(b))) \xrightarrow{Up_2^Z} ((\kappa), (B, ex), (\nu(X), \nu(Y), 1 - \frac{\beta}{2}, \nu(b)))$.

Proposition 8. For any box B and context $\langle \kappa \rangle$, and $\nu(X) = 1 - \beta_c$, we have that $((\kappa), (B, en), (1 - \beta_c, \nu(Y), \nu(Z), \nu(b))) \xrightarrow{Up_4^X} ((\kappa), (B, ex), (1 - \frac{\beta_c}{4}, \nu(Y), \nu(Z), \nu(b)))$.

Proposition 9. For any box B and context $\langle \kappa \rangle$, and $\nu(Y) = 1 - \beta_d$, we have that $((\kappa), (B, en), (\nu(X), 1 - \beta_d, \nu(Z), \nu(b))) \xrightarrow{Up_2^Y} ((\kappa), (B, ex), (\nu(X), 1 - \frac{\beta_d}{2}, \nu(Y), \nu(b)))$.

Simulate Decrement Instruction: Assume that the $(k + 1)$ st instruction is decrementing counter c . Then we construct the main component $Dec\ c$ similar to the component $Inc\ c$ above. The main change is the following:

- The main component $Dec\ c$ will have the subcomponents Up_2^Z and Up_2^Y lined up sequentially. The functionality Up_2^Z and Up_2^Y are the same as the one in $Inc\ c$. The total time spent in $Dec\ c$ is also, less than 8β .

Zero Check Instruction: The main component for Zero Check follows the same pattern as *Inc c*. The main change is the following:

- The main component *ZeroCheck* will have the subcomponents Up_2^Z and Up_2^X and Up_2^Y lined up sequentially. The functionality Up_2^Z , Up_2^X and Up_2^Y are the same as the one in *Inc c*. After these three, we invoke the subcomponent $ZC : c = 0?$ shown in Figure 7. $ZC : c = 0?$ is called by passing all stopwatches by value. At the entry node en_5 of ZC , z_1 and x_1 are ticking. At en_5 if a time elapse $\frac{\beta}{2} = \frac{1}{2^{k+1}}$ makes $z_1 = x_1 = 1$, then we know that x_1 on entry was $1 - \frac{1}{2^{c+k+1}} = 1 - \frac{1}{2^{k+1}}$ implying that $c = 0$. Likewise, if z_1 attains 1, but x_1 does not, then $c \neq 0$. Since all stopwatches are passed by value, at the return port of ZC within the main component *ZeroCheck*, we regain back the stopwatch values obtained after going through Up_2^Z , Up_2^X and Up_2^Y : that is, $\nu(Z) = 1 - \frac{\beta}{2}$, $\nu(X) = 1 - \frac{\beta_c}{2}$ and $\nu(Y) = 1 - \frac{\beta_d}{2}$. The time elapsed in ZC is $\frac{\beta}{2}$. The times elapsed in Up_2^Z and Up_2^X and Up_2^Y are same as calculated in the case of Increment c . Thus, the total time elapsed here is $< 8\beta + \beta = 9\beta$.

We conclude by calculating the total time elapsed during the entire simulation. We have established so far that for the $(k + 1)$ th instruction, the time elapsed is no more than 9β , for $\beta = \frac{1}{2^k}$. For the first instruction, the time elapsed is at most 9, for the second instruction it is $\frac{9}{2}$, for the third it is $\frac{9}{2^2}$ and so on.

$$\text{Total time duration} = \begin{cases} 9(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots) \\ 9(1 + (< 1)) \\ < 18 t.u \end{cases} \quad (2)$$

The proof that we reach the vertex *Halt* of the RHA iff the two counter machine halts follows: Clearly, the exit node of each main component iff the corresponding instruction is simulated correctly. Thus, if the counter machine halts, we will indeed reach the exit node of the main component corresponding to the last instruction. However, if the machine does not halt, then we keep going between the various main components simulating each instruction, and never reach *Halt*.

□

5 Undecidability Results with two players

For the undecidability results for reachability games, we construct a recursive automaton (timed/hybrid) as per the case, whose main components are the modules for the instructions and the counters are encoded in the variables of the automaton. In these reductions, the reachability of the exit node of each component corresponding to an instruction is linked to a faithful simulation of various increment, decrement and zero check instructions of the machine by choosing appropriate delays to adjust the clocks/variables, to reflect changes in counter values. We specify a main component for each type instruction of the two counter machine, for example \mathcal{H}_{inc} for increment. The entry node and exit node of a main component \mathcal{H}_{inc} corresponding to an instruction $[\ell_i : c := c + 1; \text{goto } \ell_k]$ are respectively ℓ_i and ℓ_k . Similarly, a main component corresponding to a

zero check instruction $[l_i: \text{if } (c > 0) \text{ then goto } \ell_k] \text{ else goto } \ell_m$, has a unique entry node ℓ_i , and two exit nodes corresponding to ℓ_k and ℓ_m respectively. The various main components corresponding to the various instructions, when connected appropriately, gives the higher level component \mathcal{H}_M and this completes the RHA \mathcal{H} . The entry node of \mathcal{H}_M is the entry node of the main component for the first instruction of M and the exit node is $Halt$. Achilles simulates the machine while Tortoise verifies the simulation. Suppose in each main component for each type of instruction correctly Achilles simulates the instruction by accurately updating the counters encoded in the variables of \mathcal{H} . Then, the unique run in M corresponds to an unique run in \mathcal{H}_M . The halting problem of the two counter machine now boils down to existence of a Achilles strategy to ensure the reachability of an exit node $Halt$ (and \smile) in \mathcal{H}_M .

5.1 Time Bounded Reachability games in Unrestricted RTA with 3 clocks

Lemma 6. *The time bounded reachability game problem is undecidable for recursive timed automata with at least 3 clocks.*

Proof. We prove that the reachability problem is undecidable for unrestricted RTA with 3 clocks. In order to obtain the undecidability result, we use a reduction from the halting problem for two counter machines. Our reduction uses a RTA with three clocks x, y, z .

We specify a main component for each instruction of the two counter machine. On entry into a main component for increment/decrement/zero check, we have $x = \frac{1}{2^{k+c}3^{k+d}}$, $y = \frac{1}{2^k}$ and $z = 0$, where c, d are the current values of the counters and k is the current instruction. Note that z is used only to enforce urgency in several vertices. Given a two counter machine, we build a 3 clock RTA whose building blocks are the main components for the instructions. The purpose of the components is to simulate faithfully the counter machine by choosing appropriate delays to adjust the variables to reflect changes in counter values. On entering the entry node en of a main component corresponding to an instruction l_i , we have the configuration $(\langle \epsilon \rangle, en, (\frac{1}{2^{k+c}3^{k+d}}, \frac{1}{2^k}, 0))$ of the three clock RTA.

We shall now present the components for increment/decrement and zero check instructions. In all the components, the variables passed by value are written below the boxes and the invariants of the locations are indicated below them.

Simulate increment instruction: Lets consider the increment instruction $\ell_i: c = c + 1; \text{goto } \ell_k$. The component for this instruction is component $Inc\ c$ given in Figure 8. Assume that $x = \frac{1}{2^{k+c}3^{k+d}}$, $y = \frac{1}{2^k}$ and $z = 0$ at the entry node en_1 of the component $Inc\ c$. To correctly simulate the increment of counter c , the clock values at the exit node ex_1 should be $x = \frac{1}{2^{k+c+2}3^{k+d+1}}$, $y = \frac{1}{2^{k+1}}$ and $z = 0$.

Let $\alpha = \frac{1}{2^{k+c}3^{k+d}}$ and $\beta = \frac{1}{2^k}$. We want $x = \frac{\alpha}{12}$ and $y = \frac{\beta}{2}$ at ex_1 . We utilise the component $Div\{a, n\}$ (with $a = x, n = 12$ and $a = y, n = 2$) to perform these divisions. Lets walk through the working of the component $Inc\ c$. As seen above, at the entry node en_1 , we have $x = \frac{1}{2^{k+c}3^{k+d}}$, $y = \frac{1}{2^k}$ and $z = 0$.

1. No time is spent at en_1 due to the invariant $z = 0$. $Div\{y, 2\}$ is called, passing x, z by value. At the call port of $A_1 : Div\{y, 2\}$, we have the same values of x, y, z . Let us examine the component $Div\{y, 2\}$. We instantiate $Div\{a, n\}$ with $a = y, n =$

2. Thus, the clock referred to as b in $Div\{a, n\}$ is x after the instantiation. At the entry node en_2 of $Div\{y, 2\}$, no time is spent due to the invariant $z = 0$; we have $a = y = \beta, b = x = \alpha, z = 0$. Resetting $b = x$, we are at the call port of $A_3 : D$. A_3 is called, passing a, z by value. A nondeterministic time t is spent at the entry node en_3 of D . Thus, at the return port of A_3 , we have $a = y = \beta, b = x = t, z = 0$. The return port of A_3 is a node belonging to Tortoise; for Achilles to reach \smile , t must be $\frac{\beta}{2}$. Tortoise has two choices to make at the return port of A_3 : he can continue the simulation, by resetting $a = y$ and going to the call port of $A_5 : D$, or he can verify if t is indeed $\frac{\beta}{2}$, by going to the call port of A_4 .

- Assume Tortoise goes to the call port of $A_4 : C_{y/2}^{x=}$ (recall, that by the instantiation, $b = x, a = y$ and $n = 2$). z is passed by value. At the entry node en_5 of $C_{y/2}^{x=}$, no time elapses due to the invariant $z = 0$. Thus, we have $x = b = t, a = y = \beta, z = 0$ at en_5 . The component $A_7 : M_x$ is invoked, passing x, z by value. At the entry node en_6 of M_b , a time $1 - t$ is spent, giving $a = y = \beta + 1 - t, b = x = t$ and $z = 0$ at the return port of A_7 . Since $n = 2$, one more invocation of $A_7 : M_x$ is made, obtaining $a = y = \beta + 2(1 - t), b = x = t$ and $z = 0$ at the return port of A_7 after the second invocation. To reach the exit node ex_5 of $C_{y/2}^{x=}$, a must be exactly 2, since no time can be spent at the return port of A_7 ; this is so since the invariant $z = 0$ at the exit node ex_5 of $C_{y/2}^{x=}$ is satisfied only when no time is spent at the return port of A_7 . If a is exactly 2, we have $\beta = 2t$. In this case, from the return port of A_4 , \smile can be reached.
- Now consider the case that Tortoise moves ahead from the return port of A_3 , resetting $a = y$ to the call port of $A_5 : D$. The values are $a = y = 0, b = x = t = \frac{\beta}{2}$ and $z = 0$. $A_5 : D$ is invoked passing $b = x$ and z by value. A non-deterministic amount of time t' is spent at the entry node en_3 of D , giving $a = y = t', b = x = \frac{\beta}{2}$ and $z = 0$ at the return port of A_5 . Again, the return port of A_5 is a node belonging to Tortoise. Here Tortoise, thus has two choices: he can continue with the simulation going to ex_2 , or can verify that $t' = \frac{\beta}{2}$ by going to the call port of $A_6 : C_x^{y=}$. $C_x^{y=}$ is a component that checks if y has “caught up” with x ; that is, whether $t' = t = \frac{\beta}{2}$. At the entry node en_4 of $C_x^{y=}$, a and b can simultaneously reach 1 iff $t = t'$; that is, $t' = \frac{\beta}{2}$. Then, from the return port of A_6 , we can reach \smile .
- Thus, we reach ex_2 with $x = y = \frac{\beta}{2}, z = 0$. At the return port of $A_1 : Div\{y, 2\}$, we thus have $x = \alpha, y = \frac{\beta}{2}, z = 0$.

2. From the return port of $A_1 : Div\{y, 2\}$, we reach the call port of $A_2 : Div\{x, 12\}$. y, z are passed by value. The functioning of A_2 is similar to that of A_1 : at the return port of A_1 , we obtain $x = \frac{\alpha}{12}, y = \frac{\beta}{2}$ and $z = 0$.

Time taken: Now we discuss the total time to reach a \smile node or the exit node ex_1 of the component Inc while simulating the increment instruction. At the entry node en_1 , clock values are $x = \frac{1}{2k+c3k+a}, y = \frac{1}{2k}$ and $z = 0$. Let $\alpha = \frac{1}{2k+c3k+a}$ and $\beta = \frac{1}{2k}$. The invariant $z = 0$ at the entry and the exit nodes en_1 and ex_1 ensures that no time elapses in these nodes and also in the return ports of A_1 and A_2 . From the analysis above, it follows that at the return port of $A_1 : Div\{y, 2\}$, $x = \alpha, y = \frac{\beta}{2}$ and $z = 0$. Similarly

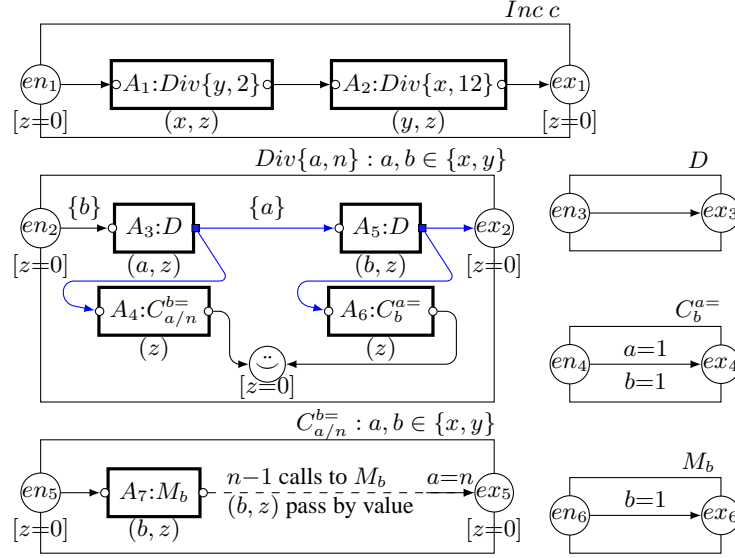


Fig. 8. Games on RTA with 3 clocks : Increment c .

at the return port of $A_2 : Div\{x, 12\}$, the clock values are $x = \frac{\alpha}{12}$, $y = \frac{\beta}{2}$ and $z = 0$. Thus, counter c has been incremented and the end of instruction k has been recorded in x and y . The time spent along the path from en_1 to ex_1 is the sum of times spent in $A_1 : Div\{y, 2\}$ and $A_2 : Div\{x, 12\}$.

- Time spent in $A_1 : Div\{y, 2\}$. The time spent in $A_3 : D$, as well as $A_5 : D$ is both $\frac{\beta}{2}$. Recall that Tortoise can verify that the times t, t' spent in A_3, A_5 are both $\frac{\beta}{2}$. If Tortoise enters A_4 to verify $t = \frac{\beta}{2}$, then the time taken is $2(1 - t)$. In this case, the time taken to reach \smile from the return port of A_4 is $t + 2(1 - t) = 2 - \frac{\beta}{2}$. Likewise, if Tortoise continued from A_3 to A_5 , and goes on to verify that the time t' spent in A_5 is also $\frac{\beta}{2}$, then the total time spent before reaching the \smile from the return port of A_6 is $t + t' + (1 - t') = 1 + t = 1 + \frac{\beta}{2}$. Thus, if we are back at the return port of A_1 , the time spent in A_1 is $t + t' = \beta$.
- Time spent in $A_2 : Div\{x, 12\}$. Here, the time spent in $A_3 : D$ as well as $A_5 : D$ is $\frac{\alpha}{12}$. In case Tortoise verifies that the time t spent in $A_3 : D$ is indeed $\frac{\alpha}{12}$, then he invokes A_4 . The time elapsed in $C_{x/12}^{y=}$ is $12(1 - t) = 12(1 - \frac{\alpha}{12}) < 12$. Likewise, if Tortoise continued from A_3 to A_5 , and goes on to verify that the time t' spent in A_5 is also $\frac{\alpha}{12}$, then the total time spent before reaching the \smile from the return port of A_6 is $t + t' + (1 - t') = 1 + t = 1 + \frac{\alpha}{12}$. Thus, if we are back at the return port of A_2 , the time spent in A_2 is $t + t' = \frac{2\alpha}{12}$.
- In general, the component $Div\{a, n\}$ divides the value in clock a by n . If $a = \zeta$ on entering $Div\{a, n\}$, then upon exit, its value is $a = \frac{\zeta}{n}$. The time taken to reach the

exit ex_2 is $2 * (\frac{\zeta}{n})$. The time taken to reach the node \smile in $Div\{a, n\}$ is $< n$ (due to n calls to M_b component).

- Total time spent in $Inc\ c$. Thus, if we come back to the return port of A_2 , the total time spent is $\beta + \frac{2\alpha}{12} < 2\beta$, on entering with $y = \beta$. Recall that $x = \alpha = \frac{1}{2^{k+c}3^{k+d}}$ and $y = \beta = \frac{1}{2^k}$ and thus $\alpha \leq \beta$ always.

From the analysis above, the following propositions are easy to see.

Proposition 10. *For any context $\kappa \in (B \times V)^*$, any box $b \in B$, and $x, y \in [0, 1]$, there exists a unique strategy of Achilles such that*

$$\begin{aligned} & (\langle \kappa \rangle, (b, en_2), (x, y = \beta, 0)) \xrightarrow[Div\{y, 2\}]{\beta} (\langle \kappa \rangle, (b, ex_2), (x, \frac{\beta}{2}, 0)), \\ & \text{or } (\langle \kappa \rangle, (b, en_2), (x, y, 0)) \xrightarrow[Div\{y, 2\}]{2 - \frac{\beta}{2}} (\langle \kappa, (b, (x, y, 0)) \rangle, \smile, (\frac{\beta}{2}, y, 0)), \quad (3) \\ & \text{or } (\langle \kappa \rangle, (b, en_2), (x, y, 0)) \xrightarrow[Div\{y, 2\}]{1 + \frac{\beta}{2}} (\langle \kappa, (b, (x, y, 0)) \rangle, \smile, (\frac{\beta}{2}, \frac{\beta}{2}, 0)). \end{aligned}$$

Proposition 11. *For any context $\kappa \in (B \times V)^*$, any box $b \in B$, and $x, y \in [0, 1]$, we have that*

$$(\langle \kappa \rangle, (b, en_1), (x, y, 0)) \xrightarrow[Inc\ c]{\leq 2\beta} (\langle \kappa \rangle, (b, ex_1), (\frac{x}{12}, \frac{y}{2}, 0)).$$

The proof essentially relies on the argument given above. Using summary edges, we can easily obtain the result.

Simulate Zero check instruction: Let us now simulate the instruction l_i : if $(d > 0)$ then goto ℓ_k , else goto ℓ_j . Figure 9 describes this. The component for this instruction is component *Zero Check* : $d = 0?$. Starting with $x = \frac{1}{2^{k+c}3^{k+d}} = \alpha$, $y = \frac{1}{2^k} = \beta$ and $z = 0$ at the entry node en_1 , we want to reach the node corresponding to ℓ_k if $d > 0$, with $x = \frac{1}{2^{k+c+1}3^{k+d+1}} = \frac{\alpha}{6}$, $y = \frac{1}{2^{k+1}} = \frac{\beta}{2}$ and $z = 0$, and to the node corresponding to ℓ_j if $d = 0$, with the same clock values. The nodes $d = 0$ and $d > 0$ are respectively the exit nodes of the component *Zero Check* : $d = 0?$. In the following, we analyse the zero check component in detail.

1. The first component invoked on entry is $A : Div\{y, 2\}$ that records the $k + 1$ th instruction by dividing y by 2. Clocks x, z are passed by value. This component is the same as seen in the *Inc c* component. As seen there, at the return port of $A : Div\{y, 2\}$, we obtain $x = \alpha$, $y = \frac{\beta}{2}$, $z = 0$. A time of β is spent in the process. Similarly, at the return port of $B : Div\{x, 6\}$, we obtain $x = \frac{\alpha}{6}$, $y = \frac{\beta}{2}$ and $z = 0$. A total time of $\frac{2\alpha}{6}$ is elapsed in $B : Div\{x, 6\}$. Thus, the total time spent on coming to the return port of $B : Div\{x, 6\}$ is $\beta + \frac{2\alpha}{6} < 2\beta$.
2. At the return port of B , we goto the node m , elapsing no time. This is needed since the exit nodes of the zero check component have the invariant $z = 0$. At m , Achilles guesses whether $d = 0$ or not, and goes to one of m_1, m_2 . Both these nodes belong to Tortoise. At both m_1, m_2 , Tortoise has two choices: he can go to an exit node of the zero check component, or choose to verify the correctness of the guess of Achilles. The \smile node is reachable from the upper component $B_1 : ZC_{=0}^d$ if $d = 0$, while \smile node is reachable from the lower component $B_1 : ZC_{=0}^d$ if $d > 0$. Lets now look at the component $ZC_{=0}^d$.

3. At the entry node en_2 , we have $x = \frac{\alpha}{6}, y = \frac{\beta}{2}$ and $z = 0$. To check if $d = 0$, we first eliminate the k from x, y , obtaining $x = 6^{k+1} \cdot \frac{\alpha}{6} = \frac{1}{2^{c3^d}}$ and $y = 2^{k+1} \cdot \frac{1}{2^{k+1}} = 1$. The component B_3 multiplies y by 2 once, and invokes B_4 , which multiplies x by 6; this is repeated until y becomes 1. B_3 is invoked passing x, z by value, while B_4 is invoked passing y, z by value. Lets examine the functioning of $Mul\{y, 2\}$, the functioning of $Mul\{x, 6\}$ is similar.
4. At the entry node en_3 of $Mul\{a, n\}$, with $a = y, n = 2$ and $b = x$, we have $z = 0, x = \frac{\alpha}{6}, y = \frac{\beta}{2}$. Resetting $b = x$, we goto the call port of $B_6 : D$. D is called passing y, z by value. A non-deterministic time t is spent at the entry node en_5 of D ; thus, at the return port of B_6 , we have $b = x = t, a = y = \frac{\beta}{2}, z = 0$. The time t must be β ; Tortoise can verify this by invoking $B_7 : C_{2*y}^{x=}$. $C_{2*y}^{x=}$ invokes M_y two times, passing y, z by value. Each time, in M_y , a time of $1 - \frac{\beta}{2}$ is spent. After the two invocations, we obtain $b = x = t + 2(1 - \frac{\beta}{2}), a = y = \frac{\beta}{2}$ and $z = 0$. This b must be exactly 2 to reach the exit node ex_4 of $C_{2*y}^{x=}$; this is possible iff $t = \beta$. In this case, Tortoise will allow Achilles to goto the \smile node from the return port of $B_6 : D$. If Tortoise skips the verification and goes directly to $B_8 : D$, then at the call port of $B_8 : D$, we have $b = x = \beta, a = y = 0$ and $z = 0$. D is called by passing $b = x, z$ by value. A time t' is elapsed in D , obtaining $b = x = \beta, a = y = t'$ and $z = 0$. This t' must be exactly β ; Tortoise can verify this by invoking $B_9 : C_x^{y=}$, at the return port of B_8 . $C_x^{y=}$ checks if y has “caught up” with x ; that is, if y is also β . Clearly, the exit node ex_6 is reached iff $a = b$; that is, $t' = \beta$. At the return port of B_8 , we thus have $a = b = \beta, z = 0$. Back at the return port of B_3 , we thus obtain $x = \frac{\alpha}{6}, y = \beta, z = 0$.
5. In a similar way, $Mul\{x, 6\}$ multiplies x by 6. This, at the return port of B_4 , y is multiplied by 2 and x by 6, once. The process repeats until we obtain $y = 1$ at the return port of B_4 . At this time, we know that the loop has happened $k + 1$ times, that is, $y = 1$ and $x = \frac{1}{2^{c3^d}}$.
6. Now we can check if d is zero or not, by multiplying x by 2^{c+1} times. If x becomes exactly 2 at sometime, then clearly d is zero; otherwise, x will never become exactly 2. Then the only option is to goto the exit node $d > 0$ of $ZC_{=0}^d$. If Achilles had guessed correctly that $d = 0$ and gone to node m_1 , in the zero check component, then $ZC_{=0}^d$ will reach the upper exit node $d = 0$; From this return port of B_1 , \smile is reachable. Similar is the case when Achilles guesses correctly that $d > 0$ at m .

Time taken:

- The total time taken to reach the return port of $B : Div\{x, 6\}$ is $\beta + \frac{\alpha}{3} < 2\beta$, on entering en_1 with $y = \beta, x = \alpha, z = 0$.
- The total time taken to reach the return port of B_3 , having entered the call port of B_3 with $y = \frac{\beta}{2}$ is $2\beta = 4\frac{\beta}{2}$. Likewise, the total time taken to reach the return port of B_4 , having entered the call port of B_4 with $x = \frac{\alpha}{6}$ is $2\alpha = 12\frac{\alpha}{6}$. Thus, the total time to reach the return port of B_4 after one round of multiplication of x, y is $4\frac{\beta}{2} + 12\frac{\alpha}{6} < 4\beta + 12\beta = 16\beta$. The second time B_3, B_4 loop is invoked is with $y = \beta, x = \alpha$, the times taken respectively will be 4β and 12α , and so on. Thus, the total time taken until y becomes 1 is $< 16(\beta + 2\beta + 2^2\beta + \dots + 1) < 16$. Recall that $x = \alpha = \frac{1}{2^{k+c3^{k+d}}}$ and $y = \beta = \frac{1}{2^k}$ and thus $\alpha \leq \beta$ always.

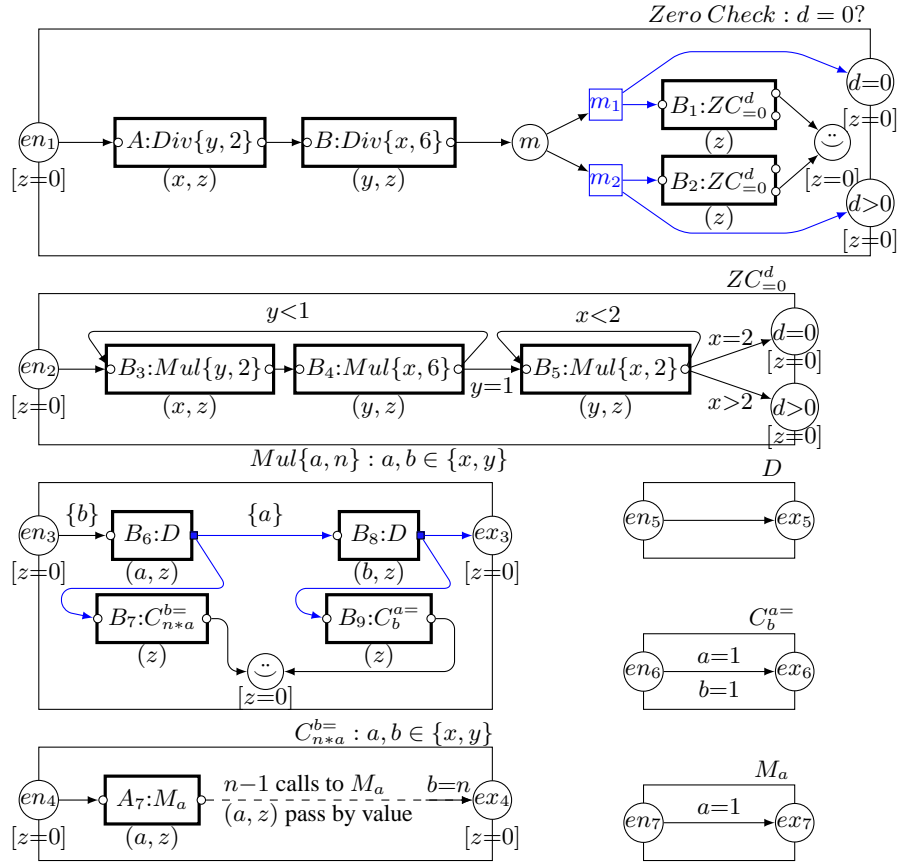


Fig. 9. Games on RTA with 3 clocks : Zero check $c = 0$?

- Once y becomes 1, the $B_5 : Mul\{x, 2\}$ loop is taken until x reaches 2 or beyond. B_5 is entered with $x = \frac{1}{2^c 3^d} = \gamma$, and B_5 is invoked $c + 1$ times. The first time $Mul\{x, 2\}$ is invoked with $x = \gamma$, the time elapsed is 2γ ; the next time $Mul\{x, 2\}$ is invoked with $x = 2\gamma$, the time elapsed is 4γ and so on. Thus, the total time elapsed in B_5 loop is $2\gamma + 2^2\gamma + \dots + 2^{c+1}\gamma < 2$, where $2^{c+1}\gamma = \frac{1}{3^d}$. If $d = 0$, then after $c + 1$ steps, the exit node $d = 0$ of $ZC_{=0}^d$ is reached; if $d > 0$, then the loop is taken $d + 2$ more times; in this case also, the total time elapsed to reach the exit node $d > 0$ is < 2 . Thus the total time taken in $ZC_{=0}^d$ component is < 16 from the $B_3 - B_4$ loop and < 2 from B_5 loop. Thus time to reach either exit of this component is < 18 .
- In general, the component $Mul\{a, n\}$ multiplies the value in clock a by n . If $a = \zeta$ on entering $Mul\{a, n\}$, then upon exit, its value is $a = n * \zeta$. The functioning of this component is very similar to that of $Div\{a, n\}$ described earlier. The time taken to reach the exit ex_3 is $2 * (n * \zeta)$. The time taken to reach the node \smile in $Mul\{a, n\}$ is $< n$ (due to n calls to M_a component in $C_{n*a}^{b=}$).
- Total time taken in *Zero Check* : $d = 0?$. Time taken to come to the return port of B is $< 2\beta$. No time is spent at the return port of B , at node m, m_1, m_2 . No time is thus spent on reaching the exit nodes $d = 0$ or $d > 0$ from the return port of B . Thus, total time taken to reach an exit node of *Zero Check* : $d = 0?$ is $< 2\beta$, on entering with $y = \beta$. The time taken to reach \smile node in this component is $< 18 + 2\beta$ where < 18 t.u is the time elapsed in component $ZC_{=0}^d$.

Other instructions: The main component to simulate other instructions are as follows.

- Decrement c : In main component *Inc c* of Figure 11, the second call $Div\{x, 12\}$ is replaced by $Div\{x, 3\}$ thus updating x from $\frac{1}{2^{k+c} 3^{k+d}}$ to $\frac{1}{2^{k+c} 3^{k+d+1}}$ to record end of k instruction.
- Increment d : $Div\{x, 12\}$ is replaced by $Div\{x, 18\}$ to update x to $\frac{1}{2^{k+c+1} 3^{k+d+2}}$.
- Decrement d : $Div\{x, 2\}$ is used to update x to $\frac{1}{2^{k+c+1} 3^{k+d}}$ recording end of k instruction.
- Zero check $c = 0?$: Call $B_5 : Mul\{x, 2\}$ is replaced by $B_5 : Mul\{x, 3\}$ and the time taken to reach the exits remains the same.

In all these cases, the time taken to reach \smile would be < 18 time units (in $Div\{x, 18\}$). Also, on entering any of the main components with $y = \beta$, an exit node is reached in $< 2\beta$ units of time.

Complete RTA : We obtain the full RTA simulating the two counter machine by connecting the entry and exit of main components of instructions according to the machine's sequence of instructions. If the machine halts, then the RTA has an exit node corresponding to *HALT*. Anytime Tortoise embarks on a check, a \smile is reachable if Achilles has simulated the instruction correctly. As observed above, on entering any component corresponding to an instruction with $y = \beta$, the exit node of that component can be reached in time $< 2\beta$, and a \smile node can be reached in time < 18 . Now the time to reach the exit node *HALT* is the time taken for the entire simulation of the machine. As Tortoise can enter any of the check components, Achilles is bound to choose the correct delays to update the counters accurately. We conclude by calculating the total time elapsed during the entire simulation. We have established so far that for

the (k) th instruction, the time elapsed is no more than 2β , for $\beta = \frac{1}{2^k}$. For the first instruction, the time elapsed is at most 2, for the second instruction it is $\frac{2}{2}$, for the third it is $\frac{2}{2^2}$ and so on.

$$\text{Total time duration} = \begin{cases} 2(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots) \\ = 2(1 + (< 1)) \\ < 4 \end{cases} \quad (4)$$

We now show that the two counter machine halts iff Achilles has a strategy to reach $HALT$ or \smile . Suppose the machine halts. Then the strategy for Achilles is to choose the appropriate delays to update the counters in each main component. Now if Tortoise does not verify (by entering check components) in any of the main components, then the exit ex_1 of the main component is reached. If Tortoise decides to verify then the node \smile (follows from Proposition 10 and 11) is reached. Thus, if Achilles simulates the machine correctly then either the $HALT$ exit or \smile is reached if the machine halts.

Conversely, assume that the two counter machine does not halt. Then we show that Achilles has no strategy to reach either $HALT$ or \smile . Consider a strategy of Achilles which correctly simulates all the instructions. Then \smile is reached only if Tortoise chooses to verify. But if Tortoise does not choose to verify then \smile can not be reached. The simulation continues and as the machine does not halt, the exit node $HALT$ is never reached. Now, consider any other strategy of Achilles which does an error in simulation (in a hope to reach $HALT$). Tortoise could verify this, and in this case, the node \smile will not be reached as the delays are incorrect. Thus Achilles can not ensure reaching $HALT$ or \smile with a simulation error.

5.2 Time bounded reachability games in RSA

Lemma 7. *The time bounded reachability game problem is undecidable for glitchfree recursive stopwatch automata with at least 4 stopwatches.*

Proof. We outline quickly the changes as compared to Lemma 6. The proof proceeds by the simulation of a two counter machine. Figure 10 gives the component for incrementing counter c . There are 4 stopwatches x, y, z, u . The encoding of the counters in the variables is similar to Lemma 6: at the entry node of each main component simulating the k th instruction, we have $x = \frac{1}{2^{c+k}3^{d+k}} = \alpha$, $y = \frac{1}{2^k} = \beta$ and $z = 0$, where c, d are the current values of the counters. We use the extra stopwatch u for rough work and hence we do not ensure that $u = 0$ when a component is entered.

Simulate increment instruction: As was the case in Lemma 6, simulation of the $(k + 1)$ th instruction, incrementing c amounts to dividing y by 2 and x by 12. In Lemma 6, it was possible to pass some clocks by value, and some by reference, but here, all variables must be either passed by value or by reference.

The $Div\{a, n\}$ module here is similar to that in Lemma 6: the box $A_3 : D$ in Figure 8 is replaced by the node l_1 , where only u ticks and accumulates a time t . (Recall that u is the stopwatch used for rough work and has no bearing on the encoding.) In node l_2 , only z ticks. l_2 is a node belonging to Tortoise. The time t spent at l_1 must be exactly $t = \frac{\beta}{2}$, where $\beta = \frac{1}{2^k}$ is the value of $a = y$ on entering $Div\{y, 2\}$. In this case, Tortoise, even if he enters the check module $C_{y/2}^u$, will reach \smile .

Again, note that the module $C_{y/2}^{u=}$ is similar to the one in Figure 8. We use $b = x$ for rough work in this component. Due to this the earlier value of x is lost. However, this does not affect the machine simulation as only \smile of $Div\{y, 2\}$ is reached and not the exit node and the simulation does not continue. At en_5 (of M_u), we have $b = x = 0$, $u = t$ and $a = y = \beta$. a, b, u tick at en_5 . A time $1 - t$ is spent at en_5 , obtaining $b = 1 - t, u = 0, a = \beta + (1 - t)$ at l . At l , only b, u tick obtaining $b = 0, u = t, a = \beta + (1 - t)$ at ex_5 . A second invocation of $C_{y/2}^{u=}$ gives $b = 0, u = t, a = \beta + 2(1 - t)$. For $a = 2$, to reach ex_3 , we thus need $t = \frac{\beta}{2}$. The time elapsed in one invocation of M_u is 1 time unit; thus a total of $2+t$ time units is elapsed before reaching \smile (via module $C_{y/2}^{u=}$ in $Div\{a, n\}$).

If Tortoise skips the check at l_2 and proceeds to l_3 resetting $a = y$, we have at l_3 , $z = 0, u = t = \frac{\beta}{2}$ and $a = 0$. Only a ticks at l_3 , a is supposed to “catch up” with u at l_3 , by elapsing $t = \frac{\beta}{2}$ in l_3 . Again, at l_4 , only z ticks. Tortoise can verify whether $a = u$ by going to $C_u^{a=}$. The component $C_u^{a=}$ is exactly same as that in Figure 8. A time of $1 - t$ is elapsed in $C_u^{a=}$. Thus, the time taken to reach \smile from $C_u^{a=}$ is $t + t + 1 - t = 1 + t$. Thus, ex_2 is reached in time $2t = 2 \cdot \frac{\beta}{2} = \beta$. As was the case in Lemma 6, the time taken to reach the exit node of $Inc\ c$, starting with $y = \beta, x = \alpha, z = 0$ is $\beta + 2 \cdot \frac{\alpha}{12} < 2\beta$. Also, the time taken by $Div\{a, n\}$ on entering with $a = \zeta$ is $2 \cdot \frac{\zeta}{n}$.

To summarize, the time taken to reach the exit node of the $Inc\ c$ component is $< 2\beta$, on entering with $y = \beta$. Also, the component $Div\{a, n\}$ divides the value in clock a by n . If $a = \zeta$ on entering $Div\{a, n\}$, then upon exit, its value is $a = \frac{\zeta}{n}$. The time taken to reach the exit ex_2 is $2 * (\frac{\zeta}{n})$. The time taken to reach the node \smile in $Div\{a, n\}$ is $< n + 1$ (due to n calls to M_u component).

Simulate Zero check instruction: Here again, we illustrate the changes as compared to the zero check done in Lemma 6. Figure 11 describes the zero check module. As in the case of Figure 9, on entering en_1 with $x = \alpha, y = \beta, z = 0$, we divide y by 2 and x by 6, to record the $(k + 1)$ th instruction in x, y . These modules are already discussed in the increment instruction above. We only discuss the module $Mul\{a, n\}$ here. This is similar to the $Div\{a, n\}$ module seen above. If we enter $Mul\{a, n\}$ with $a = \zeta$, at location l_1 , a time $t = \zeta \cdot n$ should be spent. This makes $u = \zeta \cdot n$, the values of a, z are unchanged. Tortoise can verify that $t = \zeta \cdot n$ using $C_{n*a}^{u=}$. It can be seen that the $Mul\{a, n\}$ module here is similar to the $Mul\{a, n\}$ module in Figure 9.

Complete RSA: As in the case of Lemma 6, the complete RSA is constructed by connecting components according to the machine instructions. The time elapses in the components are exactly the same as those in Lemma 6. Thus the total time duration for machine simulation is < 4 . Along the same lines, we can also prove that Achilles has a strategy to reach HALT or \smile iff the machine halts.

Lemma 8. *The time bounded reachability game problem is undecidable for unrestricted recursive stopwatch automata with at least 3 stopwatches.*

This follows from Lemma 6.

5.3 Reachability games on RSA

Reachability problem in recursive stopwatch automata with a single player is studied in Section 4. The problem is undecidable for unrestricted recursive stopwatch automata

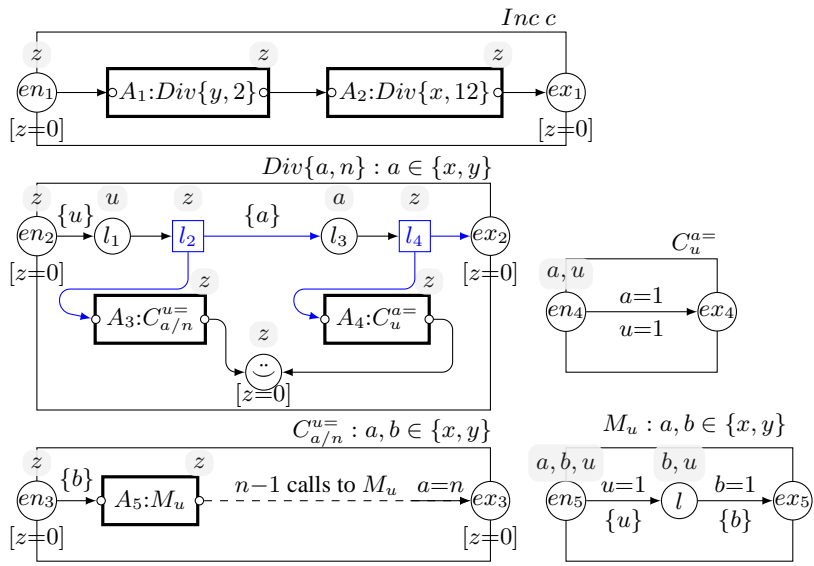


Fig. 10. Games on Glitchfree-RSA with 4 stopwatches : Increment c . Note that the variables that tick in a location are indicated above it. Due to semantics of RHA, no time elapses in the call ports and exit nodes and hence no variables ticking is not mentioned for these locations.

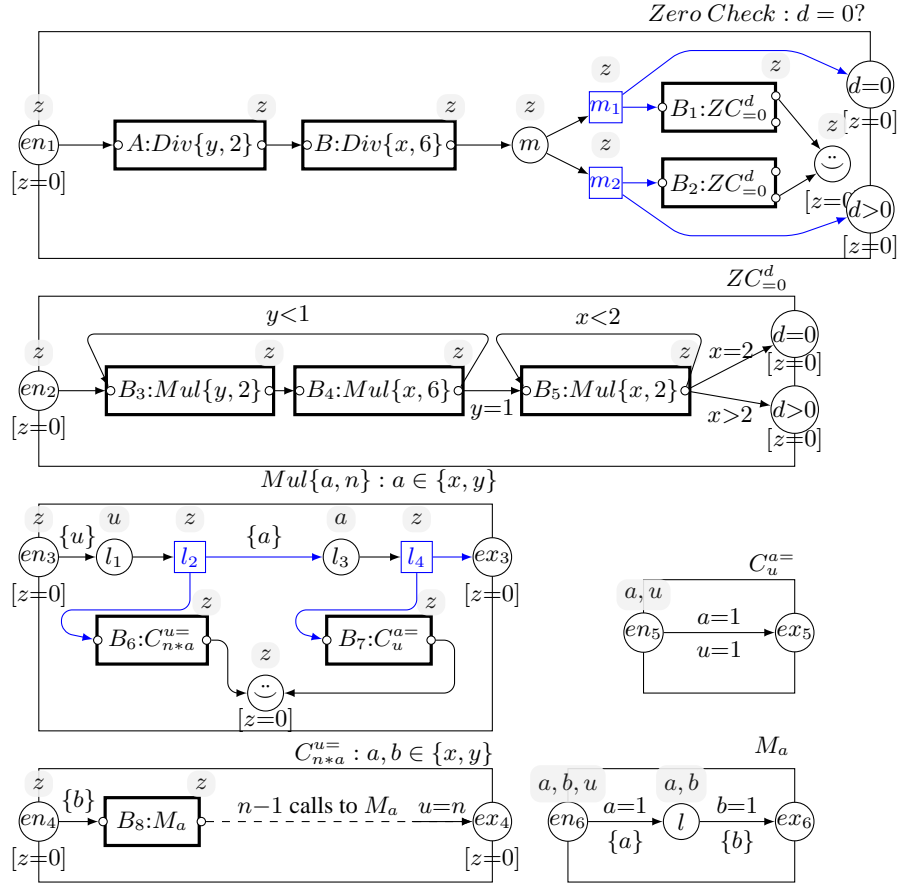


Fig. 11. Games on Glitchfree-RSA with 4 stopwatches : Zero check $c = 0$?. Note that the variables that tick in a location are indicated above it. z is ticking in all return ports of boxes $B_1 : ZC_{=0}^d$ and $B_2 : ZC_{=0}^d$ but not indicated above all of them, to avoid clutter. Due to semantics of RHA, no time elapses in the call ports and exit nodes and hence no variables ticking is not mentioned for these locations.

with atleast two stopwatches. Further, it is undecidable for the glitchfree variant with atleast 3 stopwatches. The details of these results in Sections 4.1 and 2. Due to these, following results in two player games on RSA are easy to see.

Lemma 9. *The reachability game problem is undecidable for glitchfree recursive stopwatch automata with at least 3 stopwatches.*

Lemma 10. *The reachability game problem is undecidable for unrestricted recursive stopwatch automata with at least 2 stopwatches.*

6 Decidability with one player : Bounded Context RHA using only pass-by-reference

We mainly discuss the results of Theorem 4 here;

6.1 Hybrid automata : Time bounded reachability [10]

Time bounded reachability was shown to be decidable for hybrid automata with no negative rates and no diagonal constraints [10]. The main idea here is that if there is a run ρ between two configurations (q_1, ν_1) and (q_2, ν_2) in a hybrid automata H such that $\text{duration}(\rho) \leq T$ (called T -time bounded run), then there exists a *contracted* run ρ' between the same configurations, such that $\text{duration}(\rho') \leq T$, length of ρ' is atmost C , a constant exponential in H and linear in T , and is dependent on $rmax$ (maximal rate in H) and $cmax$ (largest constant in the constraints of H). The construction of ρ' from ρ relies on a *contraction* operator. This operator identifies positions $i < j$ in ρ , such that all locations between i and j are visited before i in ρ and locations $l_i = l_j$ and $e_{i+1} = e_{j+1}$ the outgoing edges from l_i and l_j respectively. The operator then deletes all the locations $i + 1, \dots, j$ and adds their time to the other occurrences before i . It then connects $l_i \xrightarrow{e_{j+1}} l_{j+1}$ with sum of time delays accompanying e_{i+1} and e_{j+1} . This operator is used as many times as required until a fixpoint is reached. Care should be taken to ensure that the *contracted run* is a valid run : it should satisfy the constraints. To ensure this, the run ρ is first carefully partitioned into exponentially many pieces, so that contracting the pieces and concatenating them yields a valid run.

Firstly, to help track whether the valuations resulting from contraction satisfy constraints, the region information is stored in the locations to form another hybrid automaton $R(H)$. Given $cmax$, the set of regions is $\{(a - 1, a), [a, a] | a \in \{1, \dots, cmax\}\} \cup \{0^-, 0^+, (cmax, +\infty)\}$. It differs from the classical region notion due to lack of fractional part ordering (no diagonal constraints) and special treatment of valuations which are 0. $R(H)$ checks whether a variable x never changes from 0 before the next transition, or if it becomes > 0 before the next transition. This helps bound the number of sub-runs that are constructed later, and prevents the contraction operator from merging locations where x remains 0 with those where x becomes > 0 . The construction ensures that H admits a run between two states of duration T iff $R(H)$ admits a run between the same states and for the same time T .

As the rest of the automaton is untouched, the equivalent of run ρ in $R(H)$ is a run same as ρ , but having region information along with locations. Let us continue to call

the run in $R(H)$ as ρ . ρ is called a *type-0 run*. ρ is chopped into fragments of duration $\leq \frac{1}{rmax}$, each of which is called a *type-1 run*. There will be atmost $T.rmax + 1$ type-1 runs. Additionally, as $rmax$ is the maximal rate of growth of any variable, a variable changes its region atmost 3 times such that, when starting in $(b, b + 1)$ region, growing through $[b + 1, b + 1]$, $(b + 1, b + 2)$, gets reset and stays in $[0, 1)$. Each type-1 run is further split into type-2 runs based on region changes which is atmost 3 times per variable. Thus each type-1 run is split into atmost $3 \cdot |\mathcal{X}|$ type-2 runs. Respecting region changes ensures that constraints continue to be satisfied post contraction. Type-2 runs are again split into type-3 runs based on the first and last reset of a variable. This is to enable concatenation of consecutive contracted fragments by ensuring the valuations in the start configuration and end configuration of each fragment are compatible with their neighbors. Each type-2 run is split into atmost $2 \cdot |\mathcal{X}| + 1$ type-3 runs. The contraction is applied to type-3 runs, removing second occurrences of loops. Hence, each contracted type-3 run will be atmost $|Loc'|^2 + 1$ long (Lemma 7 of [10]), where Loc' is the set of locations of $R(H)$. Note that $|Loc'| = |Loc| \cdot (2.cmax + 1)^{|\mathcal{X}|}$ where Loc is the set of locations of H . After concatenating these contracted type-3 runs, we get contracted type-2 runs with the same start and end states. These contracted type-2 runs are then concatenated to obtain a run ρ' of that has the same start and end states as ρ , duration $(\rho') = \text{duration}(\rho)$ and $|\rho'| \leq C = 24 \cdot (T.rmax + 1) \cdot |\mathcal{X}|^2 \cdot |Loc|^2 \cdot (2.cmax + 1)^{2 \cdot |\mathcal{X}|}$. To solve time-bounded reachability, we nondeterministically guess a run of length at most C , and solve an LP to check if there are time delays and valuations for each step to make the run feasible.

6.2 Bounded-Context RHA with pass-by-reference only mechanism : Time bounded reachability

Along the lines of contraction operator, we define a *context-sensitive* contraction operator cnt for a run in the bounded context RHA. As seen in Section 6.1, we convert the bounded context RHA H into $R(H)$, where we remember the respective regions along with the vertices of H . In the rest of this discussion, when we say H , we mean $R(H)$.

The contraction operator in [10] matches locations in the run while we match the (context, location) pairs of the configurations in the run. The context matching ensures that we do not alter the sequence of recursive calls made in the contracted run, thus maintaining validity w.r.t recursion. The second occurrence is then deleted and the time delays are added to the first occurrence of the loop. Let us denote the (context, location) pair to be used for matching as $cl = (\langle \kappa \rangle, q)$. Ignoring the valuations, we denote a context as $\kappa \in B^*$ since all the variables are passed by reference and hence need not be stored in the context. Henceforth, we shall denote a run as $\rho = (\langle \kappa_0 \rangle, q_0, \nu_0) \xrightarrow{t_1, e_1} (\langle \kappa_1 \rangle, q_1, \nu_1) \xrightarrow{t_2, e_2} \dots (\langle \kappa_{n-1} \rangle, q_{n-1}, \nu_{n-1}) \xrightarrow{t_n, e_n} (\langle \kappa_n \rangle, q_n, \nu_n)$ where e_i is the discrete transition enabled after the time delay t_i in the vertex q_{i-1} .

Definition 3 (Context-sensitive contraction cnt). Consider a run $\rho = (\langle \kappa_0 \rangle, q_0, \nu_0) \xrightarrow{t_1, e_1} (\langle \kappa_1 \rangle, q_1, \nu_1) \xrightarrow{t_2, e_2} \dots (\langle \kappa_{n-1} \rangle, q_{n-1}, \nu_{n-1}) \xrightarrow{t_n, e_n} (\langle \kappa_n \rangle, q_n, \nu_n)$. Assume there are two positions $0 \leq i < j < n$ and a function $h : \{i + 1, \dots, j\} \rightarrow \{0, \dots, i - 1\}$ such that (i) $(\langle \kappa_i \rangle, q_i) = (\langle \kappa_j \rangle, q_j)$ and (ii) for all $i < p < j$: $(\langle \kappa_p \rangle, q_p) = (\langle \kappa_{h(p)} \rangle, q_{h(p)})$.

Then $\text{cnt}(\rho) = (\langle \kappa'_0 \rangle, q'_0, \nu'_0) \xrightarrow{t'_1, e'_1} (\langle \kappa'_1 \rangle, q'_1, \nu'_1) \xrightarrow{t'_2, e'_2} \dots (\langle \kappa'_{m-1} \rangle, q'_{m-1}, \nu'_{m-1}) \xrightarrow{t'_m, e'_m} (\langle \kappa'_m \rangle, q'_m, \nu'_m)$ where

1. $m = n - (j - i)$
2. for all $0 \leq p < i$, $(\langle \kappa'_p \rangle, q'_p) = (\langle \kappa_p \rangle, q_p)$
3. for all $1 \leq p < i$, $e'_p = e_p$ and $t'_p = t_p + \sum_{k \in h^{-1}(p-1)} t_{k+1}$
4. $e'_{i+1} = e_{j+1}$ and $t'_{i+1} = t_{i+1} + t_{j+1}$
5. for all $i+1 < p \leq m$, $(\langle \kappa'_p \rangle, q'_p) = (\langle \kappa_{p+j-i} \rangle, q_{p+j-i})$

Given a run ρ , $\text{cnt}^0(\rho) = \rho$, $\text{cnt}^1(\rho) = \text{cnt}(\rho)$, $\text{cnt}^i(\rho) = \text{cnt}(\text{cnt}^{i-1}(\rho))$. The fixpoint $\text{cnt}^*(\rho) = \text{cnt}^n(\rho)$ such that $\text{cnt}^n(\rho) = \text{cnt}^{n-1}(\rho)$. We shall prove in the following lemmas that the length of $\text{cnt}^*(\rho)$ is independent of ρ .

Lemma 11. Given a type-3 run ρ in the bounded context RHA H , $|\text{cnt}^*(\rho)| \leq (\alpha \cdot |Q| \cdot (2 \cdot \text{cmax} + 1)^{|\mathcal{X}|})^2 + 1$, where $\alpha = \sum_{i=1}^K n^i$, K is the bound on the context length, and n is the number of boxes in H .

Proof. Contraction of [10], matches the locations in a type-3 run. Thus the size of a contracted type-3 run is $|Loc|^2 + 1$ (Lemma 7 of [10]) where Loc is the set of locations in the region hybrid automata. However, we match (context, location) pairs in our context-sensitive contraction.

Suppose ρ is a type-3 run. Let $\rho' = \text{cnt}^*(\rho)$ have M unique (context, location) pairs. Highlighting the first occurrences of these M unique pairs and ignoring the valuations, we have in ρ' , $(\langle \kappa_1 \rangle, q_1) \underline{w_1} (\langle \kappa_2 \rangle, q_2) \underline{w_2} \dots (\langle \kappa_{M-1} \rangle, q_{M-1}) \underline{w_{M-1}} (\langle \kappa_M \rangle, q_M) \underline{w_M}$ where w_i are strings over (context, location) pairs, which does not have any first occurrence of a (context, location) pair. Clearly, there are M first occurrences of (context, location) pairs $(\langle \kappa_i \rangle, q_i)$ for $1 \leq i \leq M$. Let a *portion* be a part of ρ' between two such first occurrences $(\langle \kappa_{i-1} \rangle, q_{i-1})$ and $(\langle \kappa_i \rangle, q_i)$, $2 \leq i \leq M+1$. In a portion, contraction cannot be applied anymore. If it could be, then $\text{cnt}^*(\rho)$ is not a fixpoint. There could be a cl pair such that its first occurrence is at index i , and second occurrence is at index j , $i < j$; that is, $cl_i = (\langle \kappa_i \rangle, q_i) = (\langle \kappa_j \rangle, q_j) = cl_j$, and the index j is part of a later portion (cl_i is thus underlined, but cl_j is not). We cannot contract the pairs at indices i, j , since *all pairs* between cl_i and cl_j would not occur prior to cl_i (if they occur, then $\rho' \neq \text{cnt}^*(\rho)$). Thus the number of unique pairs in a portion could be more than 1 and can be atmost M . Thus the maximum length of $\rho' \leq M^2 + 1$.

There are atmost $\alpha = \sum_{i=1}^K n^i$ different contexts of size atmost K with any sequence of the n boxes including a box being called more than once. We know that M is the number of unique (context, location) pairs. Clearly, $M \leq \alpha \cdot |Q| \cdot (2 \cdot \text{cmax} + 1)^{|\mathcal{X}|}$, where $Q = \bigcup_{i=1}^n Q_i$ is the union of the set of vertices Q_i of all the n components of the RHA, $|Q| \cdot (2 \cdot \text{cmax} + 1)^{|\mathcal{X}|}$ is the number of vertices in the region RHA and K is the context bound. Thus, the length of a type-3 contracted run is $\leq (\alpha \cdot |Q| \cdot (2 \cdot \text{cmax} + 1)^{|\mathcal{X}|})^2 + 1$. Thus proved.

Let us illustrate with an example why a contracted type-3 run $|\text{cnt}^*(\rho)|$ could be of length $> \alpha \cdot |Q| \cdot (2 \cdot \text{cmax} + 1)^{|\mathcal{X}|}$. Let CL be a set of unique (context, location)

pairs ($|CL| \leq \alpha \cdot |Q| \cdot (2.cmax + 1)^{|\mathcal{X}|}$). Assume $CL = \{a, b, c, d, e, f\}$. Let us abuse notation of a run for a short while and depict it to be only a sequence of pairs ignoring the valuations. Now let $cnt^*(\rho) = \rho' = \underline{a} \rightarrow \underline{b} \rightarrow \underline{c} \rightarrow \underline{d} \rightarrow a \rightarrow \underline{e} \rightarrow b \rightarrow \underline{f}$. Here the portions are ϵ (between a and b , between b and c , and between c and d) and a (between d and e) and b (between e and f). Note that each of these portions themselves can not be contracted any further. Additionally although there are two occurrences of a (position 0 and 4) itself, the pairs between the first and second occurrence of a (these pairs are b, c, d) do not appear prior to a at position 0. Thus contraction can not be applied to ρ' . Thus $|cnt^*(\rho)|$ could be $> \alpha \cdot |Q| \cdot (2.cmax + 1)^{|\mathcal{X}|}$. However, each portion itself could be atmost $\alpha \cdot |Q| \cdot (2.cmax + 1)^{|\mathcal{X}|}$ (number of unique pairs). Thus $|cnt^*(\rho)| \leq (\alpha \cdot |Q| \cdot (2.cmax + 1)^{|\mathcal{X}|})^2 + 1$.

Lemma 12. *Given a run ρ in the bounded context RHA H , $|cnt^*(\rho)| \leq 24(T.rmax + 1)|\mathcal{X}|^2(\alpha|Q|)^2 \cdot (2.cmax + 1)^{2|\mathcal{X}|}$.*

Proof. Recall the splitting of a given run prior to contraction detailed in Section 6.1. The given run ρ (type-0) yields $(T.rmax + 1)$ type-1 runs each of which is further split into $(3 \cdot |\mathcal{X}|)$ type-2 runs. Each type-2 run is split into $2 \cdot |\mathcal{X}| + 1$ type-3 runs. Each contracted type-3 run is atmost $(\alpha \cdot |Q| \cdot (2.cmax + 1)^{|\mathcal{X}|})^2 + 1$ long (Lemma 11 above). Thus length of each contracted type-2 run is

$$\begin{aligned} &\leq [2 \cdot |\mathcal{X}| + 1] \cdot [(\alpha \cdot |Q| \cdot (2.cmax + 1)^{|\mathcal{X}|})^2 + 1] \\ &\leq 2 \cdot (|\mathcal{X}| + 1) \cdot ((\alpha \cdot |Q| \cdot (2.cmax + 1)^{|\mathcal{X}|})^2 + 1) \\ &\leq 2 \cdot (2 \cdot |\mathcal{X}|) \cdot (2 \cdot (\alpha \cdot |Q| \cdot (2.cmax + 1)^{|\mathcal{X}|})^2) = 8 \cdot |\mathcal{X}| \cdot (\alpha \cdot |Q| \cdot (2.cmax + 1)^{|\mathcal{X}|})^2 \end{aligned}$$

Thus the length of $cnt^*(\rho) \leq [(T.rmax + 1) \cdot (3 \cdot |\mathcal{X}|)] \cdot [8 \cdot |\mathcal{X}| \cdot (\alpha \cdot |Q| \cdot (2.cmax + 1)^{|\mathcal{X}|})^2]$
 $= 24(T.rmax + 1) \cdot |\mathcal{X}|^2 \cdot (\alpha \cdot |Q|)^2 \cdot (2.cmax + 1)^{2|\mathcal{X}|}$.

Lemma 13. *Given a run ρ in the bounded context RHA H , $cnt^*(\rho)$ is a valid run in H .*

Proof. To prove that the contracted run $\rho' = cnt^*(\rho)$ is valid in the given bounded context RHA H , we need to ensure two conditions :

- the constraints appearing along the transitions of ρ' are still satisfied and
- the sequence of boxes (and mapping of call, return, entry, exit vertices) is valid w.r.t recursive calls in the given RHA. This means call port and appropriate entry node should be consecutive in the contracted run, exit node-return ports are matched and the contexts in configurations of ρ' should be valid successors of the preceeding contexts.

The first condition is satisfied as we consider a variant of RHA where all the variables are always passed by reference. Thus the context has no valuations but only a sequence of boxes. The constraints are guaranteed to be satisfied as the run is same as a hybrid automata run if the context is ignored. Thus the precautions (in carefully splitting from type-0 to type-3 runs) taken in [10] for hybrid automata suffice with regards to constraints.

The second condition is satisfied due to context-sensitive contraction where in the context is also matched in the loop detection. Due to this, a context in the contracted run will be a valid successor of the preceeding context. We shall prove by contradiction that

there exists no invalid pair of consecutive configurations in the contracted run ρ' . There are several ways in which a pair of configurations can be invalid predecessor/successor w.r.t recursion:

- the call port and entry node are mismatched (either of them is missing or matched to another box's entry node)
- the return node and exit port are mismatched
- the consecutive contexts are incorrect/invalid (the sequences of boxes in the two contexts are such that it is not possible in the RHA semantics to get one sequence from another via a valid RHA move.)

Let ρ denote a run in the RHA, and let ρ' be its contraction. For ease of explanation, let's call the successor of a configuration c in ρ as $\text{succ}(c)$ and its predecessor as $\text{pred}(c)$. Suppose there exists a pair of consecutive configurations in ρ' which are invalid w.r.t recursive call. Abusing notation, we henceforth consider the context as $\kappa \in B^*$, ignoring valuations, as all variables are always passed by reference and hence need not be stored in the context.

Let us assume that the contracted run ρ' has a pair of consecutive configurations $c' \xrightarrow{t,e} d'$ which are invalid as the call port configuration c' is not succeeded by the appropriate entry node configuration in the contracted run i.e; $c' = (\langle \kappa \rangle, (b, \text{en}), \nu')$ and $d' \neq (\langle \kappa, b \rangle, \text{en}, \nu')$ ($t = 0$ by RHA semantics). Consider configurations $c = (\langle \kappa \rangle, (b, \text{en}), \nu)$ and $\text{succ}(c) = (\langle \kappa, b \rangle, \text{en}, \nu)$ in the given run ρ such that c' corresponds to c . As $d' \neq (\langle \kappa, b \rangle, \text{en}, \nu')$, the configuration $\text{succ}(c)$ was deleted during contraction. Thus it must be the case that in ρ , c was at position i while the repeated (context, location) pairs were from position $i + 1$ to j and d (corresponding to d' in ρ') was at position $j + 1$. But, the configuration at $i + 1$ is $\text{succ}(c)$ and this was matched with a configuration, say e occurring prior to c in ρ : recall the contraction operator deletes repeating occurrences of (location, context) pairs; to delete $\text{succ}(c)$ at position $i + 1$, we have to match the (location, context) pair of $\text{succ}(c)$ at position $i + 1$ with that of some e , occurring at a position $m < i$. Due to the semantics of RHA, $\text{pred}(e)$ has (context, location) pair $(\langle \kappa \rangle, (b, \text{en}))$ which is the same as c . Thus even c would be matched to $\text{pred}(e)$ and deleted. This contradicts our assumption that c' (equivalent of c) exists in ρ' .

In essence, the call port-entry node configurations always appear consecutive to each other in a given run ρ . Thus, matching a call-port configuration to a configuration c will invariably match the corresponding entry-node configuration to the $\text{succ}(c)$ which will also be the same entry-node configuration. Similarly, exit-node and corresponding return-port configurations always appear consecutive to each other.

Now, let's consider another pair of invalid consecutive configurations $c'_1 \xrightarrow{t,e} c'_2$ in ρ' such that $c'_1 = (\langle b_1 \rangle, q_1, \nu'_1)$ and $c'_2 = (\langle b_1 b_2 b_3 \rangle, q_2, \nu'_2)$. Clearly, such a sequence is invalid under the RHA semantics (even if q_1 is a call port). During context-sensitive contraction, we match the (context, location) pairs and do not alter the contexts. Now consider two configurations c_1 and c_2 in ρ which correspond to c'_1 and c'_2 respectively. Hence $c_1 = (\langle b_1 \rangle, q_1, \nu_1)$ and $c_2 = (\langle b_1 b_2 b_3 \rangle, q_2, \nu_2)$ (valuations would differ in the two paths due to contraction). Obviously c_2 is not successor of c_1 in ρ , as ρ is the given run (hence valid) and hence can't have such a pair of consecutive configurations. For

$c'_1 \xrightarrow{t,e} c'_2$ in ρ' , it must be the case that in ρ , c_1 is the i th configuration and the repeated (context, location) pairs are from $i+1$ to j and c_2 is the $j+1$ configuration. Let $\text{pred}(c_2)$ be the predecessor of c_2 in ρ , i.e; the configuration appearing at position j . Contraction deletes the configurations from $i+1$ to j , after matching the (location, context) pairs of positions i, j , and hence c'_2 ends up as successor of c'_1 . By definition 3 of contraction, since we verify $(l_i, \text{context}_i) = (l_j, \text{context}_j)$, we know that the (context, location) pair of c_1 is the same as that of $\text{pred}(c_2)$ (recall c_2 occurs at position $j+1$, hence $\text{pred}(c_2)$ is at position j). But then the context of $\text{pred}(c_2)$ is $\langle b_1 \rangle$ which can not be followed by $\langle b_1 b_2 b_3 \rangle$ of c_2 in ρ . This contradicts our assumption that c'_2 succeeds c'_1 in ρ' .

Similar proof by contradiction can be given for any pair of configurations invalid w.r.t recursion.

Theorem 5. *Time bounded reachability is decidable for bounded context RHA using only pass-by-reference mechanism.*

Proof. Contraction of a given run ρ yields a smaller run ρ' whose length is independent of ρ . From Lemma 12, we know that $|\rho'| \leq C = 24(T.\text{max}+1)|\mathcal{X}|^2(\alpha|Q|)^2(2.\text{max}+1)^{2|\mathcal{X}|}$. Additionally, from Lemma 13, ρ' is a valid run in the RHA. Thus a non-deterministic algorithm (as in the case of Hybrid automata [10]) can be used to guess a run of length atmost C and then solve an LP to check if there are time values and valuations for each step that make such a run feasible.

6.3 Unbounded context RHA with pass-by-reference only mechanism

We show that our adaptation of contraction does not work on RHAs with unbounded context, as in figure 12. We can not apply context-sensitive contraction as the context might grow unboundedly and matching pairs may not be found within a type-3 run.

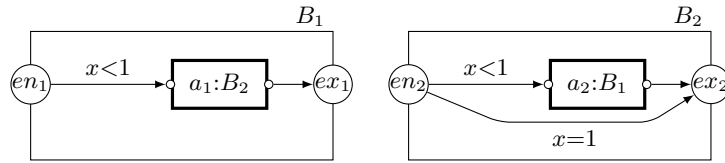


Fig. 12. RHA with unbounded context

Consider the run $\rho = (\langle \varepsilon \rangle, en_1, 0) \xrightarrow{0.1} (\langle \varepsilon \rangle, (a_1, en_2), 0.1) \xrightarrow{0} (\langle a_1 \rangle, en_2, 0.1) \xrightarrow{0.1} (\langle a_1 \rangle, (a_2, en_1), 0.2) \xrightarrow{0} (\langle a_1 a_2 \rangle, en_1, 0.2) \xrightarrow{0.1} (\langle a_1 a_2 \rangle, (a_1, en_2), 0.3) \xrightarrow{0} (\langle a_1 a_2 a_1 \rangle, en_2, 0.3) \xrightarrow{0.1} (\langle a_1 a_2 a_1 \rangle, (a_2, en_1), 0.4) \xrightarrow{0} (\langle a_1 a_2 a_1 a_2 \rangle, en_1, 0.4) \xrightarrow{0.1} (\langle a_1 a_2 a_1 a_2 \rangle, (a_1, en_2), 0.5) \xrightarrow{0} (\langle a_1 a_2 a_1 a_2 a_1 \rangle, en_2, 0.5) \xrightarrow{0.5} (\langle a_1 a_2 a_1 a_2 a_1 \rangle, ex_2, 1) \xrightarrow{0} (\langle a_1 a_2 a_1 a_2 \rangle, (a_1, ex_2), 1) \xrightarrow{0} (\langle a_1 a_2 a_1 a_2 \rangle, ex_1, 1) \xrightarrow{0} (\langle a_1 a_2 a_1 \rangle, (a_2, ex_1), 1) \xrightarrow{0}$

$(\langle a_1 a_2 a_1 \rangle, ex_2, 1) \xrightarrow{0} (\langle a_1 a_2 \rangle, (a_1, ex_2), 1) \xrightarrow{0} (\langle a_1 a_2 \rangle, ex_1, 1) \xrightarrow{0} (\langle a_1 \rangle, (a_2, ex_1), 1) \xrightarrow{0}$
 $(\langle a_1 \rangle, ex_2, 1) \xrightarrow{0} (\langle \varepsilon \rangle, (a_1, ex_2), 1) \xrightarrow{0} (\langle \varepsilon \rangle, ex_1, 1). \text{duration}(\rho) = 1.$

Now consider another run

$\rho' = (\langle \varepsilon \rangle, en_1, 0) \xrightarrow{0.1} (\langle \varepsilon \rangle, (a_1, en_2), 0.1) \xrightarrow{0} (\langle a_1 \rangle, en_2, 0.1) \xrightarrow{0.9} (\langle a_1 \rangle, ex_2, 1) \xrightarrow{0}$
 $(\langle \varepsilon \rangle, (a_1, en_2), 1) \xrightarrow{0} (\langle \varepsilon \rangle, ex_1, 1).$

Note that the start and end configurations of both the runs are the same and $\text{duration}(\rho') = 1$. However, we can not apply context-sensitive contraction in this case. There can be several other runs like ρ which can have unbounded contexts but have the same effect as ρ' . To be able to obtain ρ' from ρ , we would need to apply contraction to the contexts too and shorten them in a sensible manner. Our context-sensitive contraction studied earlier does not alter the contexts and hence does not readily extend to this class of RHA. We conjecture this to be decidable with a double layered contraction - one altering the contexts and the other altering the valuations (as seen in [10]).

7 Decidability with one player : Glitch-free RHA with 2 stopwatches

7.1 Region Abstraction of Hybrid Automata with 2 Stopwatch Variables

We first show that the reachability problem is decidable for stopwatch automata (hybrid automata with only stopwatches) with 2 stopwatch variables.

Definition 4 (Singular Hybrid Automata). A Singular Hybrid automaton is a tuple $H = (Q, Q_0, \Sigma, X, \Delta, I, F)$ where

- Q is a finite set of control modes including a distinguished initial set of control modes $Q_0 \subseteq Q$,
- Σ is a finite set of actions,
- V is an (ordered) set of variables,
- $\Delta \subseteq Q \times \text{rect}(V) \times \Sigma \times 2^X \times Q$ is the transition relation,
- $I : Q \rightarrow \text{rect}(V)$ is the mode-invariant function, and
- $F : Q \rightarrow \mathbb{Q}^{|V|}$ is the mode-dependent flow function characterizing the rate of each variable in each mode.

Recollect that, $\text{rect}(V)$ is the set of rectangular constraints over V .

Let H be a singular hybrid automata with two stopwatch variables $V = \{x, y\}$ and as both variables are stopwatches, $F : Q \rightarrow \{0, 1\}^2$. Let c_{max} be the maximum constant used in any of the guards of H . For simplicity, we assume that the hybrid automata does not have location invariants.

Regions and Region Automaton We consider a finite partitioning \mathcal{R} of \mathbb{R}^2 . For each valuation $\nu = (\nu(x), \nu(y)) \in \mathbb{R}^2$, the unique element of \mathcal{R} that contains ν is called a region, denoted $[\nu]$. We define the successors of a region R , $\text{Succ}_{(r_x, r_y)}(R) \subseteq \mathcal{R}$, in the following natural way: For $r_x, r_y \in \{0, 1\}$,

$$R' \in \text{Succ}_{(r_x, r_y)}(R) \text{ if } \exists \nu \in R, \exists t \in \mathbb{R} \text{ such that } [\nu + (r_x, r_y)t] = R'$$

Denote by $\nu + (r_x, r_y)t$, the valuation $(\nu(x) + r_x t, \nu(y) + r_y t)$. We say that such a finite partition is a *set of regions* whenever the following condition holds:

$$R' \in Succ_{(r_x, r_y)}(R) \text{ iff } \forall \nu \in R, \exists t \in \mathbb{R} \text{ such that } [\nu + (r_x, r_y)t] = R'$$

The only kind of updates we consider are those where we reset variables to 0. A reset res maps a region R to the region $res(R)$ obtained from R by assigning value 0 to all variables which were reset to 0. The set of regions \mathcal{R} is compatible with resets res if whenever a valuation $\nu' \in R'$ is reachable from a valuation $\nu \in R$ after a reset, then R' is reachable from any $\nu \in R$ by the same reset. Formally, we have

$$R' \in res(R) \rightarrow \forall \nu \in R, \exists \nu' \in R' \text{ such that } \nu' \in res(\nu)$$

The guards φ considered in H are boolean combinations of $x \bowtie c$ where $x \in V$ and $c \in \mathbb{N}$ and $\bowtie \in \{<, >, \leq, \geq, =\}$. A region R is compatible with φ iff for all valuations $\nu \in R$, either $\nu \models \varphi$ or $\nu \models \neg \varphi$.

We first construct a set of regions for 2 stopwatch automata that are compatible with resets and guards. For $z \in \{x, y\}$, we define the set of intervals

$$\mathcal{I}_z = \{[c] \mid 0 \leq c \leq c_{max}\} \cup \{(c, c+1) \mid 0 \leq c < c_{max}\} \cup \{(c_{max}, \infty)\}$$

Define $\alpha = ((I_x, I_y), \prec)$, where \prec is a total preorder on $V_0 = \{x \in V \mid I_x \text{ is an interval of the form } (c, c+1)\}$. The region associated with α denoted R_α is the set of valuations

$$\{\nu \in \mathbb{R}^2 \mid \nu(x) \in I_x, \nu(y) \in I_y \text{ and } [(x, y \in V_0, x \prec y) \leftrightarrow (frac(\nu(x)) \leq frac(\nu(y)))]\}$$

The finite set \mathcal{R} of all such regions R_α forms a partition of \mathbb{R}^2 .

Lemma 14. *\mathcal{R} as defined above, is a set of regions.*

Proof. In the sequel, we show that \mathcal{R} is a set of regions. Consider $\alpha = ((I_x, I_y), \prec)$. If $I_x = ((c_{max}, \infty), (c_{max}, \infty))$, then for all $\nu \in R_\alpha$, for all $t \in \mathbb{R}$, $\nu + (r_x, r_y)t \in R_\alpha$ for $r_x, r_y \in \{0, 1\}$. Hence $Succ_{(r_x, r_y)}(R_\alpha) = R_\alpha$. If $Succ_{(r_x, r_y)}(R_\alpha) \neq R_\alpha$, then there is atleast one another region in $Succ_{(r_x, r_y)}(R_\alpha)$ different from R_α . Let C_α denote the region that is closest to region to R_α . Such a closest region is such that $C_\alpha \in Succ_{(r_x, r_y)}(R_\alpha)$, and for all $\nu \in R_\alpha$, for all $t \in \mathbb{R}$, if $\nu + (r_x, r_y)t \notin R_\alpha$, then $\exists t' \leq t$ such that $\nu + (r_x, r_y)t' \in C_\alpha$. Such a region $C_\alpha = ((I'_x, I'_y), \prec')$ is characterized as follows: Let $Z = \{z \in V \mid I_z \text{ is of the form } [c]\}$.

1. If $Z \neq \emptyset$ and $r_x = r_y = 1$. Then

–

$$I'_z = \begin{cases} I_z & \text{if } z \notin Z, \\ (c, c+1) & \text{if } z \in Z \text{ and } 0 \leq c < c_{max} \\ (c_{max}, \infty) & \text{if } z \in Z \text{ and } I_z = [c_{max}] \end{cases}$$

– $x \prec' y$ if $I_x = [c]$ with $0 \leq c < c_{max}$ and I'_y is of the form $(d, d+1)$.

2. If $Z \neq \emptyset$ and atleast one of r_x, r_y is 0. Then

$$I'_z = \begin{cases} I_z & \text{if } r_z = 0, \\ [c+1] & \text{if } z \notin Z \text{ and } r_z = 1 \\ (c, c+1) & \text{if } z \in Z, r_z = 1 \text{ and } 0 \leq c < c_{max} \\ (c_{max}, \infty) & \text{if } z \in Z, r_z = 1 \text{ and } I_x = [c_{max}] \end{cases}$$

– $x \prec' y$ if $r_x = 1$ and $x \in Z, y \notin Z$. If $(r_x = 0 \text{ and } x \in Z)$ or $(r_x = 1 \text{ and } x \notin Z)$ or if $x, y \in Z$, then $V'_0 = \emptyset$.

3. If $Z = \emptyset$ and $r_x = r_y = 1$. Let M denote the set of variables with the maximum fractional part, whose interval is of the form $(c, c+1)$ for $0 \leq c < c_{max}$. Then

$$I'_z = \begin{cases} I_z & \text{if } z \notin M, \\ [c+1] & \text{if } z \in M \text{ and } I_z = (c, c+1) \text{ with } 0 \leq c < c_{max} \end{cases}$$

– One variable moves to an integer value, or both variables are in (c_{max}, ∞) . Hence $V'_0 = \emptyset$.

4. If $Z = \emptyset$ and atleast one of r_x, r_y is 0. Then

$$I'_z = \begin{cases} I_z & \text{if } r_z = 0, \\ [c+1] & \text{if } z \in M, r_z = 1 \text{ and } I_z = (c, c+1) \text{ with } 0 \leq c < c_{max} \\ [c+1] & \text{if } z \notin M \text{ and } r_z = 1 \text{ and } I_z = (c, c+1) \text{ with } 0 \leq c < c_{max} \end{cases}$$

– $x \prec' y$ is same as $x \prec y$ when $r_x = r_y = 0$. Otherwise, one of the variables gets an integer value, and hence $V'_0 = \emptyset$.

We now claim that

$$\forall \nu \in \alpha, \exists t \in \mathbb{R} \text{ such that } \nu + t \in C_\alpha$$

Let ν be a valuation in α . Then let $frac(\nu(x))$ denote the fractional part of $\nu(x)$. Similarly for $\nu(y)$.

1. If $Z \neq \emptyset$ and $r_x = r_y = 1$. Let $\tau = \min\{1 - frac(\nu(z)) \mid I_z \text{ is of the form } (c, c+1)\}$. Then $\nu + (1, 1)\frac{\tau}{2}$ is in the region C_α .
2. If $Z \neq \emptyset$ and atleast one of r_x, r_y is 0.
 - If $x \in Z, y \notin Z$ and $r_x = 1$, then pick $\tau = frac(\nu(y))$. Then $\nu + (1, 0)\frac{\tau}{2}$ is in the region C_α .
 - If $r_x = 0$ and $x \in Z$, then pick $\tau = 1 - frac(\nu(y))$. Then $\nu + (0, 1)\tau$ is in the region C_α .
 - If $r_x = 1$ and $x \notin Z$, then pick $\tau = 1 - frac(\nu(x))$. $\nu + (1, 0)\tau$ is in the region C_α .
 - If $x, y \in Z$, and $r_x = 1$, then pick $\tau = 0.5$. Then $\nu + (1, 0)\tau$ is in the region C_α .
3. If $Z = \emptyset$ and $r_x = r_y = 1$.
 - Pick the variable $z \in M$. Let $\tau = 1 - frac(\nu(z))$. Then $\nu + (1, 1)\tau$ is in the region C_α .
4. If $Z = \emptyset$ and atleast one of r_x, r_y is 0.
 - If $r_x = 1$ and $r_y = 0$. Pick $\tau = 1 - frac(\nu(x))$. Then $\nu + (1, 0)\tau$ is in the region C_α .

Thus we obtain that $C_\alpha \in Succ_{(r_x, r_y)}(R_\alpha)$ is the closest successor of R_α . Inducting on C_α , we get the closest successor of C_α , which is the successor of R_α , 2 steps away, and so on. We write $R_\alpha \rightarrow_n R_\alpha^n$ if R_α^n is the n th closest successor of R_α with respect to some choice of rates (r_x, r_y) . This clearly means that there is a sequence of regions $R_\alpha^0, R_\alpha^1, R_\alpha^2, \dots, R_\alpha^n$ such that $R_\alpha^0 = R_\alpha$, and R_α^{i+1} is the closest successor of R_α^i for all $1 \leq i < n$.

In this way, we can find all successors R'_α of R_α such that $R'_\alpha \in Succ_{(r_x, r_y)}(R_\alpha)$ iff for all $\nu \in R_\alpha$ there exists some $t \in \mathbb{R}$ such that $\nu + (r_x, r_y)t \in R'_\alpha$. Hence, \mathcal{R} is indeed a set of regions partitioning \mathbb{R}^2 . \square

Given two valuations $\nu_1, \nu_2 \in R_\alpha$ for some region R_α , we say that ν_1 and ν_2 are equivalent if they lie in the same region, i.e., $[\nu_1] = [\nu_2]$.

Lemma 15. *\mathcal{R} is compatible with the guards φ and with the resets res .*

Proof. 1. Let $R' \in res(R)$. Consider $\nu_1, \nu_2 \in R$, i.e., $[\nu_1] = [\nu_2]$. Clearly, $\nu_1(x)$ and $\nu_2(x)$ lie in the same interval; same with $\nu_1(y)$ and $\nu_2(y)$. If the operation res resets x , then $res(\nu_1) = (0, \nu_1(y))$ and $res(\nu_2) = (0, \nu_2(y))$. Since $\nu_1(y)$ and $\nu_2(y)$ are in the same interval, we have $[res(\nu_1)] = [res(\nu_2)]$. Similar results are obtained when y is reset, or when both x, y are reset.

2. Let $[\nu_1] = [\nu_2]$ be valuations in the same region R . Let φ be a guard. The result can be proved by structural induction on $|\varphi|$. If φ is atomic of the form $x \sim c$, clearly, $\nu_1 \models \varphi$ iff $\nu_2 \models \varphi$, since ν_1 and ν_2 are equivalent. Assume for guards of size $\leq n - 1$. It can be seen that the inductive hypothesis can be easily extended to guards of size n .

Thus, \mathcal{R} is a finite set of regions compatible with guards and resets, partitioning \mathbb{R}^2 . \square

Hence, we can use the region abstraction for the above set of regions to obtain a region automaton $H_{\mathcal{R}}$ capturing the untimed language of H . The set of states of such a region automaton is the set $Q \times \mathcal{R}$, where Q is the set of modes of H . The initial location of $H_{\mathcal{R}}$ is $(q_0, (0, 0))$ where q_0 is the initial mode of \mathcal{H} . The transitions of $H_{\mathcal{R}}$ are defined as $(q, R) \xrightarrow{a} (q', R')$ iff there is a region \hat{R} and a transition from q to q' on (φ, a, res) in \mathcal{H} such that

- $\hat{R} \in Succ_{(r_x, r_y)}(R)$. Here r_x, r_y are the rates of variables x, y at the state q of \mathcal{H} ,
- For all $\nu \in \hat{R}$, $\nu \models \varphi$, and
- $res(\hat{R}) = R'$

The final states of the region automaton are the states (f, R) such that f is a final state of \mathcal{H} . It can be seen that the language accepted by this region automaton is indeed the untimed counterpart of $L(H)$. We thus have, the following result.

Theorem 6. *The reachability problem for hybrid automata with two stopwatch variables is decidable.*

The decidability result above extends when we consider hybrid automata with location invariants as well.

7.2 Region Abstraction for Glitchfree RHA with two stopwatch variables

Given an *RHA* \mathcal{H} with two stopwatch variables x, y , we define the regional equivalence relation $\Upsilon_R \subseteq S_{\mathcal{H}} \times S_{\mathcal{H}}$ in the following way: For configurations $s = (\langle \kappa \rangle, q, \nu)$ and $s' = (\langle \kappa' \rangle, q', \nu')$, we have $(s, s') \in \Upsilon_R$, or equivalently, $[s] = [s']$ if $q = q'$, $[\nu] = [\nu']$ and $[\kappa] = [\kappa']$ such that $\kappa = (b_1, \nu_1)(b_2, \nu_2) \dots (b_n, \nu_n)$ and $\kappa' = (b'_1, \nu'_1)(b'_2, \nu'_2) \dots (b'_n, \nu'_n)$ are such that $b_i = b'_i$ and $[\nu_i] = [\nu'_i]$.

A relation $B \subseteq S_{\mathcal{H}} \times S_{\mathcal{H}}$ defined over the set of configurations of a recursive stopwatch automaton is called a *time abstract bisimulation* if for every pair of configurations $s_1, s_2 \in S_{\mathcal{H}}$ such that $(s_1, s_2) \in B$, for every timed action $(t, a) \in A_{\mathcal{H}}$ such that $X_{\mathcal{H}}(s_1, (t, a)) = s'_1$, there exists a timed action $(t', a) \in A_{\mathcal{H}}$ such that $X_{\mathcal{H}}(s_2, (t', a)) = s'_2$, and $(s'_1, s'_2) \in B$.

Lemma 16. *Regional equivalence relation for 2 stopwatch glitch-free recursive automata is a time abstract bisimulation.*

Proof. Let us fix two configurations $s = (\langle \kappa \rangle, q, \nu)$ and $s' = (\langle \kappa' \rangle, q', \nu')$ such that $[s] = [s']$ and a timed action $(t, a) \in A_{\mathcal{H}}$ such that $X_{\mathcal{H}}(s, (t, a)) = s_a = (\langle \kappa_a \rangle, q_a, \nu_a)$. We have to find a (t', a) such that $X_{\mathcal{H}}(s', (t', a)) = s'_a = (\langle \kappa'_a \rangle, q'_a, \nu'_a)$ such that $[s_a] = [s'_a]$. There are three cases:

1. The state q is a call port. That is, $q = (b, en) \in \text{Call}$. In this case, $t = 0$, the context $\langle \kappa_a \rangle = \langle \kappa, (b, \nu) \rangle$, $q_a = en$ and $\nu_a = \nu$. Since $[s] = [s']$, we know $q' = (b, en)$ is also a call port. For $t' = 0$, and $\langle \kappa'_a \rangle = \langle \kappa', (b, \nu') \rangle$, $q'_a = en$ and $\nu'_a = \nu_a$. It is clear that $[s_a] = [s'_a]$.
2. The state q is an exit node. That is, $q = ex \in \text{EX}$. Let $\langle \kappa \rangle = \langle \kappa_*, (b, \nu_*) \rangle$ and let $(b, ex) \in \text{Ret}$. In this case, $t = 0$, the context $\langle \kappa_a \rangle = \langle \kappa_* \rangle$ and $q_a = (b, ex)$ and $\nu_a = \nu[P(b) := \nu_*]$. Now let $\langle \kappa' \rangle = \langle \kappa'_*, (b, \nu'_*) \rangle$. Again, since $[s] = [s']$, we have $q = q' = ex$, $[\kappa_*] = [\kappa'_*]$ and hence $t' = 0$, $\langle \kappa'_a \rangle = \langle \kappa'_* \rangle$ and $\nu'_a = \nu'[P(b) := \nu'_*]$. We have to show that $[\nu_a] = [\nu'_a]$.
 - $P(b) = V$. In this case, $\nu_a = \nu_*$ and $\nu'_a = \nu'_*$. Since we know that $[\nu_*] = [\nu'_*]$, we obtain $[\nu_a] = [\nu'_a]$.
 - $P(b) = \emptyset$. In this case, $\nu_a = \nu$ and $\nu'_a = \nu'$ and since $[\nu] = [\nu']$, we obtain $[\nu_a] = [\nu'_a]$.
3. If state q is of any other kind, then the result follows by the region equivalence of 2 stopwatch automata (Theorem 6).

The proof is now complete. \square

Lemma 16 allows us to extend the concept of regions abstraction to two stopwatch glitch free recursive automata.

Region Abstraction for 2 StopWatch Glitchfree RHA Let $\mathcal{H} = (V, (\mathcal{H}_1, \dots, \mathcal{H}_1))$ be a glitch-free two stopwatch RHA, where each \mathcal{H}_i is a tuple $(N_i, \text{EN}_i, \text{EX}_i, B_i, Y_i, A_i, X_i, P_i, \text{Inv}_i, E_i, J_i, F_i)$. The region abstraction of \mathcal{H} is a finite RSM $\mathcal{H}^{RG} = (\mathcal{H}_1^{RG}, \mathcal{H}_2^{RG}, \dots, \mathcal{H}_k^{RG})$ where for each $1 \leq i \leq k$, component $\mathcal{H}_i^{RG} = (N_i^{RG}, \text{EN}_i^{RG}, \text{EX}_i^{RG}, B_i^{RG}, Y_i^{RG}, A_i^{RG}, X_i^{RG})$ consists of:

- a finite set of $N_i^{RG} \subseteq (N_i \times \mathcal{R})$ of nodes such that $(n, R) \in N_i^{RG}$ if $R \models Inv(n)$. Moreover, N_i^{RG} includes the set of entry nodes $EN_i^{RG} \subseteq EN_i \times \mathcal{R}$ and exit nodes $EX_i^{RG} \subseteq EX_i \times \mathcal{R}$;
- a finite set $B_i^{RG} = B_i \times \mathcal{R}$ of boxes;
- boxes-to-components mapping $Y_i^{RG} : B_i^{RG} \rightarrow \{1, 2, \dots, k\}$ is such that $Y_i^{RG}(b, R) = Y_i(b)$. To each $(b, R) \in B_i^{RG}$, we associate a set of call ports $Call^{RG}(b, R)$ and a set of return ports $Ret^{RG}(b, R)$:
 - $Call^{RG}(b, R) = \{(((b, R), en), R') \mid R' \in \mathcal{R} \text{ and } en \in EN_{Y_i(b)}\}$, and
 - $Ret^{RG}(b, R) = \{(((b, R), ex), R') \mid R' \in \mathcal{R} \text{ and } ex \in EX_{Y_i(b)}\}$
 Let $Call_i^{RG}$ and Ret_i^{RG} be the set of call and return ports of component \mathcal{H}_i^{RG} . We write $Q_i^{RG} = N_i^{RG} \cup Call_i^{RG} \cup Ret_i^{RG}$ for the vertices of the component \mathcal{H}_i^{RG} .
- $A_i^{RG} \subseteq \mathbb{N} \times A_i$ is the set of actions such that if $(h, a) \in A_i^{RG}$, (h is the number of region hops before taking a), then $h \leq 4^{c_{max}}$, where c_{max} is the maximum constant appearing in the guards;
- a transition function $X_i^{RG} : Q_i^{RG} \times A_i^{RG} \rightarrow Q_i^{RG}$ with the natural condition that call ports and exit nodes do not have any outgoing transitions. Also, for $q, q' \in Q_i^{RG}$, $(h, a) \in A_i^{RG}$, we have that $q' = X_i^{RG}(q, (h, a))$ if one of the following is true:
 - $q = (n, R) \in N_i^{RG}$, there is a region R_a such that $R \rightarrow_h R_a$, $R_a \models E_i(n, a)$, and
 - * If $q' = (n', R')$, then $R' = R_a[J_i(a) := 0]$ and $X_i(n, a) = n'$.
 - * If $q' = (((b, R'), en), R'')$, then $R = R' = R'' = R_a[J_i(a) := 0]$ and $X_i(n, a) = (b, en)$
 - $q = (((b, R_{old}), ex), R_{now})$ is a return port of \mathcal{H}_i^{RG} . Let $R = R_{old}$ if $P_i(b) = V$ and $R = R_{now}$ otherwise. There exists a region R_a such that $R \rightarrow_h R_a$ and $R_a \models E_i((b, ex), a)$ and
 - * If $q' = (n', R')$, then $R' = R_a[J_i(a) := 0]$ and $X_i(n, a) = n'$
 - * If $q' = (((b, R'), en), R'')$, then $R' = R'' = R_a[J_i(a) := 0]$ and $X_i(n, a) = (b, en)$.

The following lemma is a direct consequence of Lemma 16 and the region abstraction for 2 stopwatch glitchfree RHAs.

Lemma 17. *Reachability (termination) problems and games on glitch-free two stopwatch RHA can be reduced to solving reachability (termination) problems and games, respectively, on the corresponding region abstraction \mathcal{H}^{RG} .*

7.3 Computational Complexity

The complexity for 2 stopwatch glitch free RHAs is the same as those of 2 clock glitch free RTAs.

8 Conclusion

The main result of this paper is that time-bounded reachability problem for recursive timed automata is undecidable for automata with five or more clocks. We also showed

that for recursive hybrid automata the reachability problem turns undecidable even for glitch-free variant with three stopwatches, and the corresponding time-bounded problem is undecidable for automata with 14 stopwatches. Using the similar proof techniques we have also studied reachability games on recursive hybrid automata, and showed that time-bounded reachability games are undecidable over recursive timed automata with three clocks. Similarly, for glitch-free recursive hybrid automata with three stopwatches time-bounded reachability games are undecidable.

References

1. Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jari Stenman. Dense-timed pushdown automata. In *LICS*, pages 35–44, 2012.
2. R. Alur, M. Benedikt, K. Etessami, P. Godefroid, T. Reps, and M. Yannakakis. Analysis of recursive state machines. *ACM Transactions on Programming Languages and Systems*, 27:786–818, July 2005.
3. R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems I*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer-Verlag, 1993.
4. R. Alur and D. Dill. Automata for modeling real-time systems. In *Proc. ICALP’90*, volume 443 of *LNCS*. Springer, 1990.
5. Rajeev Alur, Thao Dang, Joel Esposito, Rafael Fierro, Yerang Hur, F Ivančić, Vijay Kumar, Insup Lee, Pradyumna Mishra, George Pappas, and Oleg Sokolsky. Hierarchical hybrid modeling of embedded systems. In *Embedded Software*, pages 14–31. Springer, 2001.
6. Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Analysis of recursive state machines. In *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 207–220. Springer Berlin Heidelberg, 2001.
7. Thomas Ball and Sriram K. Rajamani. Bebop: A symbolic model checker for boolean programs. In *SPIN*, pages 113–130. Springer-Verlag, 2000.
8. M. Benerecetti, S. Minopoli, and A. Peron. Analysis of timed recursive state machines. In *Temporal Representation and Reasoning (TIME)*, pages 61–68, Sept 2010.
9. Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR’97*, pages 135–150, 1997.
10. Thomas Brihaye, Laurent Doyen, Gilles Geeraerts, Joël Ouaknine, Jean-François Raskin, and James Worrell. Time-bounded reachability for monotonic hybrid automata: Complexity and fixed points. In *ATVA*, pages 55–70, 2013.
11. K. Cerans. *Algorithmic Problems in Analysis of Real-time System Specifications*. PhD thesis, University of Latvia, 1992.
12. S. Chaudhari. Subcubic algorithms for recursive state machines. In *POPL*, pages 159–169, 2008.
13. Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient algorithms for model checking pushdown systems. In *Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 232–247. Springer Berlin Heidelberg, 2000.
14. Kousha Etessami. Analysis of recursive game graphs using data flow equations. In *VM-CAI’04*, pages 282–296, 2004.
15. David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.
16. T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What’s decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94 – 124, 1998.

17. Salvatore La Torre, P. Madhusudan, and G. Parlato. The language theory of bounded context-switching. *LATIN*, pages 96–107, 2010.
18. Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.
19. James Rumbaugh, Ivar Jacobson, and Grady Booch, editors. *The Unified Modeling Language Reference Manual*. Addison-Wesley Longman Ltd., Essex, UK, UK, 1999.
20. Ashutosh Trivedi and Dominik Wojtczak. Recursive timed automata. In *Automated Technology for Verification and Analysis*, volume 6252 of *Lecture Notes in Computer Science*, pages 306–324. Springer Berlin Heidelberg, 2010.
21. Igor Walukiewicz. Pushdown processes: Games and model checking. In *International Conference on Computer Aided Verification, CAV 1996*, pages 62–74, 1996.